Multi-party WebRTC Videoconferencing using Scalable Video: From Best-Effort Over-the-top to Managed Value-Added Services

by

Riza Arda Kirmizioglu

A Dissertation Submitted to the
Graduate School of Sciences and Engineering
in Partial Fulfillment of the Requirements for
the Degree of

Master of Science

in

Electrical and Electronics Engineering

KOÇ UNIVERSITY

April 30, 2019

Multi-party WebRTC Videoconferencing using Scalable Video: From Best-Effort Over-the-top to Managed Value-Added Services

Koç University

Graduate School of Sciences and Engineering

This is to certify that I have examined this copy of a master's thesis by

Riza Arda Kirmizioglu

and have found that it is complete and satisfactory in all respects,
and that any and all revisions required by the final
examining committee have been made.

Committee Members:

____
Prof. Dr. A. Murat Tekalp

____
Prof. Dr. Oznur Ozkasap

____
Asst. Prof. Dr. Ali Cengiz Begen

Date: ______________________

To the magnificence and colorful world of engineering itself with the passion of Rock'N'Roll..

ABSTRACT

At present, multi-party WebRTC videoconferencing can be configured in mesh or selective forwarding unit (SFU) architectures over the best-effort Internet. Scalable video coding (SVC) enables efficient service to peers with heterogenous network connections and terminals. In the mesh architecture each peer sends a separate video stream to each other peer where peer upload bandwidth capacity is not enough as the number of peers increase, and the peer with least download bandwidth capacity becomes limiting factor for the rest of the peers because of single non-SVC encoder. SVC enabled SFU architecture, where each peer sends a single upstream video to a central SFU, requires high capacity servers with high bandwidth network connection and the SFU becomes a single point of failure. Also, SFU architecture has more network resource consumption and end-to-end delay performance comapring to mesh architecture. To these effects, we propose one SVC enabled best-effort streaming architecture for mesh connection and three network service provider (NSP) managed SVC enabled multi-party WebRTC architectures with i) mesh, ii) distributed edge-SFU and iii) software-defined network (SDN) assisted IP multicast. Proposed best-effort mesh connection technique solves limiting overall service quality effect of least receiving peer by embedding motion-adaptive SVC. Yet, best-effort videoconferencing services require very quick responsiveness for variant network conditions. Proposed NSP-managed architectures solves this basic problem by bandwidth allocation for WebRTC services. NSP-Managed mesh architecture has the best service quality comparing to proposed best-effort and default WebRTC mesh implementations, however, it still suffers from required high upload bandwidth capacity of the peers. Distributed edge-SFU architecture introduces a novel SFU streaming type by using multiple SFU servers in a single videoconferencing session. NSP-managed distributed

edge-SFU architecture reduces total network resource consumption and end-to-end service delay considerably comparing to default SFU implementation. Nonetheless, this architecture is dependent on distributed service deployment over edge networks. SDN-assisted IP-multicast architecture is a completely new streaming type for multi-party WebRTC services. This architecture overcomes limited uplink capacity, service deployment, and single point of failure drawbacks while having better network resource consumption and the best possible end-to-end service delay performance.

ÖZETÇE

Şu anda, çok partili WebRTC video konferansı, en iyi erişim gösteren İnternet üzerinden, örgü (mesh) ve seçici yönlendirme birimi (SFU) mimarilerinde yapılandırılır. Ölçeklenebilir video kodlaması (SVC), heterojen ağ bağlantıları ve farklı erişim cihazlarına sahip eşler için verimli hizmet sağlar. Her bir eşin diğer eşlere ayrı bir video akışı gönderdiği örgü mimarisinde veri gönderme bant genişliği kapasitesi yeterli değildir. Ayrıca SVC olmayan tek bir kodlayıcı kullanılması sebebiyle veri indirme hızı en düşük olan eş diğer eşlerin hizmet kalitesini kısıtlayıcı bir etki yaratır. SVC kullanılan SFU mimarisinde her eş tek bir video akışını merkezi bir SFU sunucusuna gönderir fakat SFU sunucuları yüksek bant genişliği olan ağ bağlantısına ihtiyaç duyar ve yapısından ötürü tek arıza noktası haline gelir. Ayrıca SFU mimarisi örgü mimarisine kıyasla hem İnternet altyapı kaynaklarını daha fazla kullanır hem de uçtan uca gecikme miktarı daha fazladır. Bunlara bağlı olarak SVC kullanılan bir tane en iyi erişimi gösteren İnternet ile örgü video akış mimarisi ve üç tane de İnternet hizmet sağlayıcısı ile yönetilen çok partili WebRTC mimarisi önerdik: i) örgü, ii) dağıtılmış uç (edge) SFU ve iii) yazılım tabanlı ağ destekli çoklu internet protokolüne yayın (IP multicast). Önerilen en iyi erişim gösteren internet tabanlı örgü mimarisi düşük indirme hızı ile genel hizmet kalitesini kısıtlayan eş sorununu örgü mimarisine hareket uyumlu SVC'yi gömerek çözmektedir. Ancak en iyi erişim gösteren İnternet tabanlı görüntülü toplantı hizmetletleri değişen ağ koşullarına çok hızlı uyabilen bir yapıyı gerektirmektedir. İnternet hizmet sağlayıcısı tarafından yönetilen örgü mimamirisi önerilen en iyi erişim göstern İnternet tabanlı ve halihazırdaki örgü mimarisine kıyasla daha iyi hizmet kalitesi sağlamaktadır, lakin hâlâ daha yüksek kapasiteli veri gönderme bant genişliği gerektirmektedir. Dağıtılmış uç SFU mimarisi tek bir görüntülü konuşma oturumunda birden fazla SFU sunucusu kullanarak yeni bir SFU

veri akışı tipi sunmaktadır. İnternet hizmet sağlayıcısı ile yönetilen dağıtılmış uç SFU mimarisi toplam ağ altyapı kaynak kullanımını ve uçtan uca hizmet gecikmesini halihazırda kullanılan SFU uygulamasına kıyasla önemli miktarda düşürmektedir. Buna rağmen bu mimarinin gerçeklenmesi uç ağlarda dağıtılmış hizmet tanzim edilmesine dayanmaktadır. Yazılım tabanlı ağ destekli çoklu internet protokolüne yayın mimarisi tamamen yeni bir çok partili WebRTC yayın akışı mimarisidir. Bu mimari kısıtlı veri gönderme bant genişliği, dağıtılmış hizmet tanzim edilmesi ve tek arıza noktası sorunlarını giderirken daha az İnternet altyapı kaynaklarının kullanılmasını sağlar ve olabilecek en iyi uçtan uca hizmet gecikme kabiliyeti gösterir.

ACKNOWLEDGMENTS

First of all, I would like to thank to dear Professor A. Murat Tekalp for giving me chance to work with him which was very funny, productive, educational, and visionary. I learnt a lot from Professor Tekalp and thanks to him, I always enjoyed it. Secondly, I am very thankful and grateful to my grandfather Muzaffer Basoglu and my grandmother Muzeyyen Basoglu for all the financial and moral support. I could not come this far without them that is for sure. I hope, they will always be proud of me. Also, I would like to thank to my grandfather Riza Kirmizioglu, my grandmother Betul Kirmizioglu and my family Semih, Cemile and Busra for their support. I owe Tolga Bagci so much, he was my technical mentor, helped me a lot. Aylin Erdogan you were always with me on this way, thank you for every single piece of effort you put for me and being there for me. Umuralp Kaytaz and Gurkan Kartal, my old friends, brothers, there are tremendous things and stories to tell, so glad I have you. I am also grateful to my close friend Atakan Isbakan who helped me a lot and with whom I had so much fun. Special thanks to Gonca Bakar for the times we worked and had fun together. Thanks to Serkan Sulun, we extremely laughed while watching all those absurd comedy movies together. Thanks to Ogun Kirmemis for funny moments while we share our common Internet subculture content. Cem Eteke and Dogancan Kebude, mates, thank you for the great times we had together and funny moments with full of intellectual jokes. Thanks to Selin Yilmaz for always being positive and smiling to me. I also would like to thank to Kemal, Tugtekin, Berker, Seyhan, Nusrah, Zana for being such good mates. Finally, I would like to thank to Lemmy Kilmister, Motorhead, Rock In Peace.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

INTRODUCTION

Video streaming is one of the most important features of the web and mobile applications over the Internet. There are three main classes of video streaming schemes, namely video on demand (VoD), live streaming, and real-time communications (RTC). VoD does not require strict delay constraint for service quality, while live streaming restricts latency under around 40 seconds. Unlike live streaming, RTC is interactive, thus, it requires much more strict delay constraint that is no more than 150 milliseconds. This thesis focuses on different solutions for the RTC problem.

1.1 Motivation

This thesis is concerned about multi-party WebRTC videoconferencing services featuring various connectivity models using scalable video coding. We advocate for SDN-assisted NSP-managed service architectures for guaranteed video quality with minimum service delay, as well as providing most efficient use of network resources and even distribution of traffic load over the network. Scalable video coding, multi-party WebRTC streaming architectures, managed video services over software-defined 5G networks are the main ingredients of this thesis which are discussed in Section 1.2 in detail.

Creating peer-to-peer communication over the Internet can address this restricted delay problem, however, this is applicable for one-to-one video calls. Multi-party videoconferencing architectures introduce new challenges for RTC can be listed as: multiple receivers in different regions, usage of single video encoder due to limited CPU power of the devices, limited upload bandwidth capacity, unstable network

condition, service deployment and overload, total network resource consumption and end-to-end delay.

WebRTC is one of the most popular browser-to-browser video RTC protocol over the Internet. WebRTC videoconferencing applications can easily be implemented for web and mobile applications via its simple APIs. Thus, we decided to work on the core of the WebRTC services in order to have impact for all WebRTC applications.

State of the art multi-party WebRTC solutions are mesh and selective forwarding unit (SFU) architectures. Both architectures overcome specific multi-party videoconferencing problems, nonetheless, introduce major drawbacks. We overcome the drawbacks by introducing novel streaming architectures and improving the existing ones. Section 1.2.1 discusses state of the art WebRTC protocol and multi-party streaming architectures in detail.

Scalable video coding (SVC) produces video stream which consists of substreams. Each substream is put into different network packets, therefore, video can be streamed with different bitrates even though there is single video stream. Each bitstream can add up the overall video quality in terms of spatial resolution, temporal resolution (frame per second), or signal quality. WebRTC uses SVC enabled VP9 codec which employs spatial and temporal scalibility. SVC VP9 is discussed in Section 1.2.2.

Today OTT solutions highly used over the Internet, however, they are quite sensitive to variant network condition for RTC solutions. NSP-managed solutions improve the quality of service, the quality of experience and network resource consumption by controlling the flow orchestration and bandwidth reservation for specific services. Section 1.2.3 discusses the advantages of NSP-managed video services.

1.2 Background

1.2.1 WebRTC Protocol and Service Architectures

WebRTC consists of an IETF protocol specification and Javascript APIs defined in a series of W3C documents [4]. WebRTC APIs allow media capture, and sharing media and data streams between browsers. WebRTC uses Session Traversal Utilities

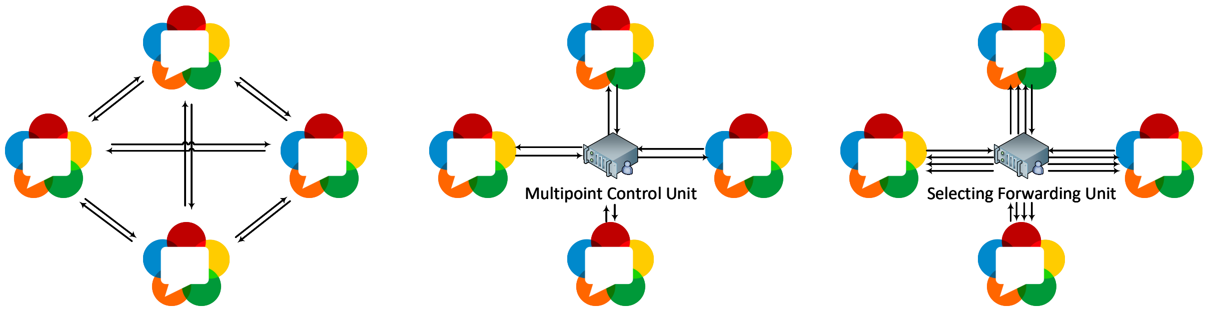

Figure 1.1: Multi-party WebRTC service architectures: (a) Mesh topology, (b) MCU topology, (c) SFU topology.

for NAT (STUN) servers to help end-points find each other in the presence of network address translation (NAT), which maps a group of private IP addresses to a single public IP address, and use Traversal Using Relays around NAT (TURN) servers as relays in the presence of firewalls.

A signaling server establishes connections between peers and creates media channels. Signaling methods and protocols are not specified by WebRTC. WebRTC media streams are delivered point-to-point using the real-time transport protocol (RTP) over the user datagram protocol (UDP). Real-time transport control protocol (RTCP) is used to monitor and estimate the available bandwidth between the end points. WebRTC platform [28] employs the Google congestion control algorithm [5] for rate control of the VP9 video encoder.

WebRTC sessions can be between two or more peers. There are three common multi-party service topologies: Mesh, MCU, and SFU service. The first two support non-scalable video coding as default, while the SFU service is specifically developed for the case of scalable video coding.

Mesh Topology

Mesh topology is the direct extension of point-to-point videoconferencing between two peers to three or more endpoints, where each peer sends separate video streams to other peers as depicted in Fig. 1.1(a). Mesh topology is advantageous in terms of

low delay, since direct connections require the smallest number of total hops between endpoints. However, it uses the limited upload bandwidth of clients inefficiently in a multi-party session. Furthermore, since each peer runs a single non-scalable video encoder (in the default configuration), the video encoding rate (quality) is determined according to the peer with the lowest receiving bandwidth.

MCU Service

MCU is a special video server (typically a cloud service) that acts as bridge between endpoints. Each peer sends video bitstream to the MCU. The MCU performs transcoding and/or mixing of incoming streams according to the requirements of the session and sends bitstreams to each peer according to their requirements. The use of an MCU avoids the upload bandwidth bottleneck at peers, and allows for more flexibility as each peer may employ a different codec and an encoding rate that is tuned to its upload bandwidth. However, processing requirements at the MCU introduces additional delay which is critical for real-time communications.

SFU Service

SFU is a special router that allows selective forwarding of different layers of scalable encoded video to different peers according to a specified configuration for a WebRTC session. Each peer runs a scalable video encoder and sends all layers of encoded video to the SFU. Unlike the MCU, there is no transcoding or processing delay at the SFU, since it only forwards packets without processing the payload. The SFU solves the peer uplink bandwidth bottleneck without introducing additional processing delay; however, it requires high speed routers with high bandwidth network connections, and like MCU the SFU becomes a single point of failure, since all peers must send streams to the same SFU.

1.2.2 Scalable VP9 Video Coding

VP9 is an open and royalty-free video compression standard developed by the WebM project [6]. VP9 has been shown to outperform H.264/AVC in terms of rate distortion performance at the expense of slightly higher computational complexity [7]. Yet, it is possible to perform real-time VP9 encoding/decoding at 30 Hz on a standard laptop for high definition (HD) video using libvpx codec implementation [19].

In addition to better compression efficiency, VP9 offers support for mixed temporal and spatial scalable video coding (SVC). A scalable encoder produces a hierarchy of encoded layers within the same bit stream. VP9 manages the spatial scalability structure using super frames, which consist of one or more layer frames, representing different spatial resolutions. In the flexible payload format mode, it is possible to change layer hierarchy and dependencies dynamically. In the non-flexible mode, the layer hierarchy and dependencies within the group of frames (GOF) are pre-specified as part of the scalability structure (SS) data. Rate adaptation can be achieved by discarding one or more spatial and/or temporal layers per frame at clients, the SFU or SDN switches.

1.2.3 NSP-Managed Video Services over SDN

Software-defined networking (SDN) introduces a new paradigm that separates the control and data planes and centralizes the control function [8]. The controller communicates with all switches by means of a southbound interface, such as OpenFlow, to specify or modify flow tables. SDN provides a practical framework for dynamic QoS provisioning since the controller has an overall view of all network resources, including switch/queue states and traffic load.

Although video traffic is increasing continuously over the years, this is not contributing to the revenues of the network service providers (NSP). NSPs can address this problem by offering value-added managed-quality video services. Managed video services refer to those services where the end-to-end bitrate and delay per flow can be centrally controlled. WebRTC combined with SDN provides an important oppor-

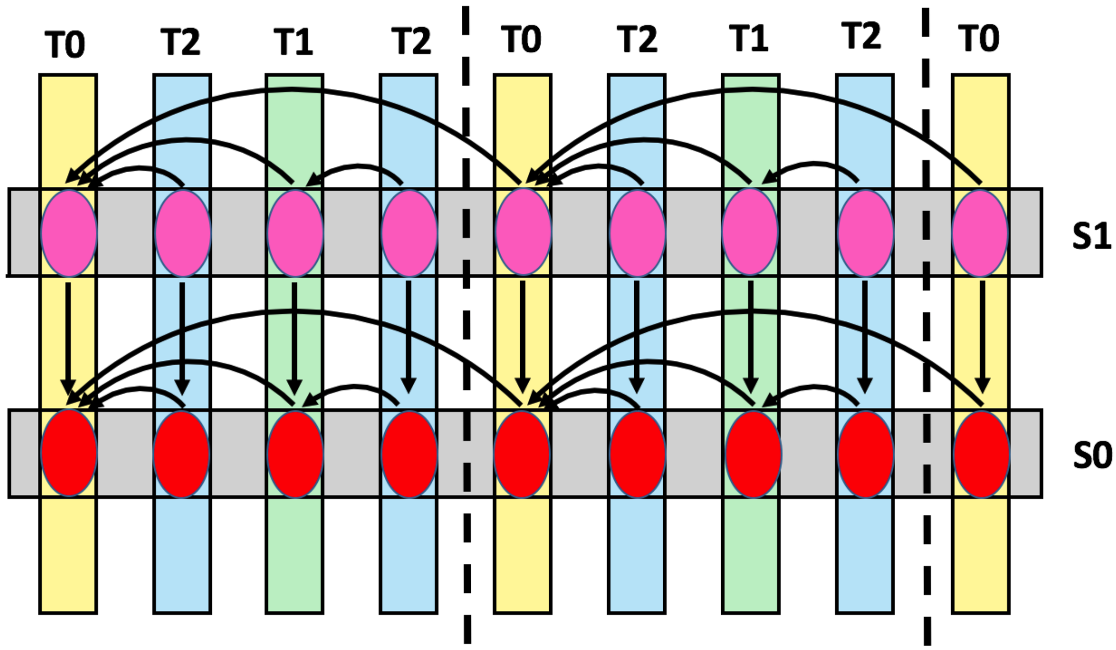

Figure 1.2: Mixed spatio-temporal scalable coding with two spatial (S0, S1) and three temporal (T0, T1, T2) layers and the dependency between the layers.

tunity for NSPs to offer value-added managed real-time communications services to increase their revenues.

1.3 Organization and Contributions of the Thesis

There are two alternative solution to default mesh architecture implementation for both OTT and managed value-added service solutions in the paper [1] which is available in Chapter 2. In OTT alternative solution, scalable video coding scheme is embedded to mesh streaming architecture in order to solve the limiting receiver downlink capacity problem by enabling flexible tunable video rate for different receivers with a single encoder. In managed value-added service solution, software defined-networking infrastructure is introduced for multi-party WebRTC application in order to enable guaranteed quality videoconferencing. The latter alternative is superior to both OTT alternative and default WebRTC solutions. Nonetheless, there is still one major drawback of mesh architecture which is limited uplink capacities of peers.

State of the art solution for multi-party WebRTC services is Selective Forwarding Unit (SFU) that employs single SVC video upstream. However, default randomly placed SFU architecture introduces huge amount of extra delay and network bandwidth resource consumption comparing to mesh architecture. Deployment of distributed SFU servers to SDN-assisted edge networks over 5G infrastructure with multiple SFU in a single videoconferencing session scheme named as Distributed Edge-SFU is proposed in the paper [2] which is available in Chapter 3.

Proposed edge-SFU architecture has much better delay and network resource consumption performance comparing to state of the art SFU implementation. Yet, proposed architecture requires deployment of the system to all edge networks and stability, since, it is single point of failure. SDN-assisted IP-multicast streaming architecture over 5G networks is proposed in the paper [3] which is available in Chapter 4. This architecture has much better network bandwidth resource consumption performance than state of the art SFU and mesh architectures and the best possible delay performance while solving limited peer uplink capacity, service deployment and maintanance, single point of failure problems.

The conclusions are presented in Chapter 4. The main contributions of this thesis are:

- Scalable video coding enabled mesh streaming architecture with motion-adaptive layer selection for best-effort OTT service that overcomes receiver limiting factor for single encoder.

- Managed multi-party WebRTC services framework over SDN for SVC-embedded mesh connected peers.

- Constrained Shortest Path Algorithm in order to find the most bandwidth available between two ends.

- Managing WebRTC services at edge networks for guaranteed quality of service.

- SFU service optimization at edge networks to minimize end-to-end delay and overall network resource usage.
- A novel distributed SFU framework, which decreases delay and avoids single point of failure.
- Development of a novel WebRTC streaming architecture using WebRTC-compliant SDN-assisted IP multicasting of scalable video for a distributed implementation of SFU functionality at network switches.
- A new bandwidth-aware and tunable delay constrained multicast tree construction algorithm.
- We propose a novel managed WebRTC services framework over SDN, where the service manager reserves bandwidth between all clients according to their device type and the rates agreed by the clients.
- Emulation environments and implementation details for proposed architectures.
- Simulation environments and implementation details of random network topologies and random service architectures for large scale service and network performance measure.

Chapter 2

MESH-CONNECTED WEBRTC VIDEOCONFERENCING SERVICES

2.1 Introduction

WebRTC has become a popular protocol for real-time communications (RTC) over the Internet that allows browser-to-browser voice, video, and data communications using simple Javascript APIs. VP9 is a royalty-free video codec that allows for efficient scalable video coding. They are both supported by the leading browsers such as Firefox and Chrome.

WebRTC videoconferencing can be offered over the current Internet as a best-effort over-the-top (OTT) service. However, RTC services often suffer from network bandwidth variations due to their stringent low-delay requirement; hence, must employ powerful network estimation and video encoding rate adaptation schemes. Alternatively, RTC with predictable and stable video quality can be offered as a managed value-added service over software-defined networks (SDN) that allows for quality of service per flow. SDN is one of the central themes of the upcoming 5G standard.

This paper proposes architectures and implementations for high-quality, browser-based, best-effort or value-added WebRTC videoconferencing services using scalable VP9 video coding. The main novelties of this paper are:

- We employ scalable video coding at the clients with motion-adaptive layer selection to match the video rate to available bandwidth in the best-effort service model,

- We propose a novel managed WebRTC services framework over SDN, where

the service manager reserves bandwidth between all clients according to their device type and the rates agreed by the clients.

We note that the proposed motion-adaptive scalable video layer selection enables clients to deal with network congestion in the most effective way in the best effort service model, while it enables clients to reserve different mesh connection bandwidths to different clients using a single encoder in the managed service model.

The rest of the paper is organized as follows: Related works are discussed in Section 2.2. The proposed best-effort WebRTC service model with motion-adaptive scalable VP9 video coding is presented in Section 2.3. Section 2.4 presents managed WebRTC services over SDN as a future value-added service model. Implementation and evaluation are discussed in Section 2.5 and Section 2.6 presents conclusions.

2.2 Related Works

We review works related to the main contributions of this paper, namely, scalable video coding in videoconferencing and WebRTC services over SDN.

A proprietary multi-party videoconferencing system using scalable video coding was first proposed in [9] in order to avoid transcoding at the multi-point control unit (MCU). Alternatively, a selective forwarding unit (SFU) can be used with scalable video coding, which selectively forwards layer streams received from clients without processing them. Scalable coding in videoconferencing is now a mainstream technology, but layer selection typically does not take video content into account. In this paper, we propose a motion-adaptive layer selection method for multi-party WebRTC using scalable VP9 coding, which extends [10] on point-to-point WebRTC videoconferencing. In multi-party videoconferencing, in addition to effectively dealing with temporary network congestion, scalable video coding allows each client to possibly select a different transmission rate to multiple mesh-connected clients using a single encoder.

Other recent WebRTC based multi-party videoconferencing solutions include P2P-MCU architecture, where the MCU functionality is hosted in one of the client peers

[11] and implementation of the MCU or SFU in the cloud [12].

Prior work on managed multi-party WebRTC services over SDN is very limited. Launching WebRTC services in SDN environment was discussed in a concept paper on "network as a service" [13], which did not have an implementation. Implementation of dynamic-network-enabled RTC on a proof-of-concept 5G network was discussed in [14].

2.3 Best-Effort OTT WebRTC Services

This section extends the motion-adaptive layer selection method for point-to-point videoconferencing presented in [10] to multi-party videoconferencing with mesh connections.

Multi-party Videoconferencing with Mesh Connections: We assume that there are N peers, and each peer generates $N-1$ RTCPeerConnection objects for mesh communication. In our implementation, we use the WebRTC native source-code which is available at [28], and the VP9 video codec with mixed spatio-temporal scalable coding option.

The rationale for using scalable video coding: We wish to run a single encoder instance at clients, since running more than one encoder causes the CPU of clients to become a bottleneck and degrade the RTC experience of users. Using a single non-scalable encoder and duplicating output video stream is also undesirable, since this restricts the flexibility of adapting the video send rate to different clients differently, and the client with the least receiving capacity shall become the limiting factor for send rate to all clients. Thus, scalable video coding at the clients offers the following advantages:

- Clients run a single encoder instance,
- Clients can adapt the send rate to different clients differently by possibly sending different layers to different clients depending on their terminal type, and

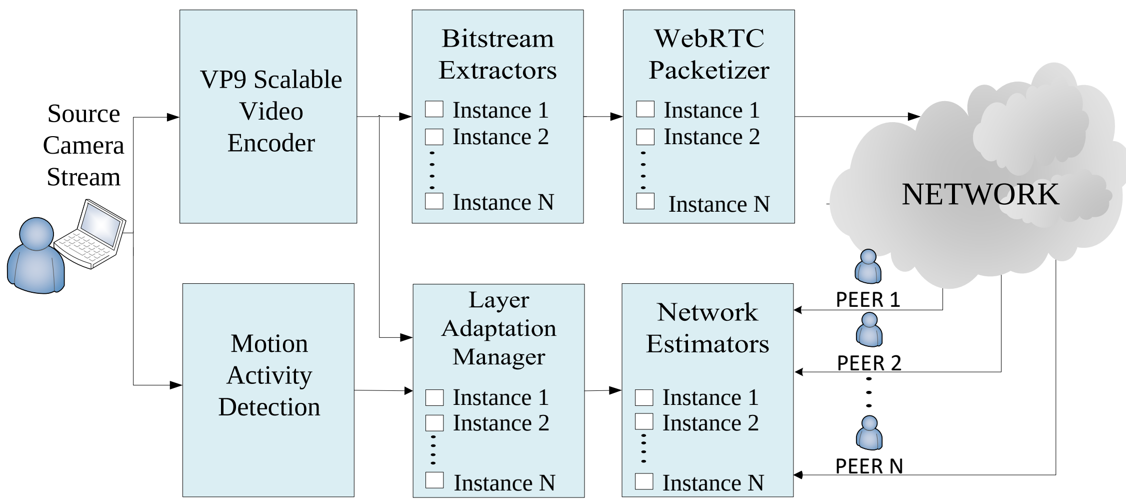

Figure 2.1: Client architecture for best-effort mesh-connected multi-party WebRTC Service

- Clients can effectively deal with bandwidth variations to different clients by temporarily dropping layers at different times depending on network conditions.

Although spatial scalable video coding has an additional overhead compared to non-scalable video coding, VP9 scalable coding comes with the smallest overhead among other scalable codecs. Nonetheless, the flexibility provided by scalable coding in multi-party videoconferencing is highly desirable.

Mixed spatio-temporal scalable coding provides two options to control video send rate at clients: i) Dropping spatial enhancement layers to reduce the spatial resolution, which typically is visually unnoticeable when there is motion; ii) Dropping temporal layers to reduce the frame rate, which typically is visually unnoticeable when there is little motion. The idea is then to make a decision between dropping spatial or temporal layers depending on the video content.

The architecture of a client is depicted in Fig. 2.1. A short description of each module follows. We note that each client runs a single instance of scalable video encoder and motion activity detection modules as singleton objects.

Network Estimator: Each clients runs a different instance of network estimation module for each PeerConnection object. The network estimator [5] uses RTCP (Real-

time Transport Control Protocol) for available bandwidth estimation.

Motion Activity Detection: Each client runs a single instance of the motion activity detection nodule. The algorithm used at each client is the same as that in [10].

Mixed Spatio-Temporal Layer Adaptation Manager: For each PeerConnection object, there is a Layer Adaptation Manager instance that receives inputs from the Motion Activity Detection and Network Estimator modules in order to adapt the video source rate to available network bandwidth by choosing which spatial or temporal will be discarded or retained. The layer adaptation algorithm is the same as in [10].

WebRTC Packetizer: For each PeerConnection object, there is a WebRTC packetizer that takes bitstream extractor output as input, packetizes the video bitstream and sends them to the network.

2.4 Managed WebRTC Services over SDN

This section introduces a novel framework for managed multi-party WebRTC videoconferencing as a value-added service, where users pay an extra fee to get guaranteed network service. The proposed service architecture is presented in Section 2.4.1. Sections 2.4.2 and 2.4.3 explain QoS routing and queue management at switches, respectively, by the SDN controller.

2.4.1 System Architecture

A traditional RTC video service provider (VSP) does not collaborate with the NSP; thus, offers a best effort service with no control over the network. In the proposed managed WebRTC services, we assume that the VSP collaborates with the NSP, and the NSP has means to offer per-flow end-to-end quality of service (QoS) by computing paths between clients with specified bandwidth and delay parameters, and implementing them by queue management at switches.

An architecture for collaboration between the RTC-VSP and the NSP to provide such a managed WebRTC service, where the NSP operates over an SDN, is depicted

in Figure 2.2. In the proposed architecture, the NSP runs a WebRTC service manager (WSM), a traffic engineering manager (TEM), and a WebRTC policing unit (WPU) over the SDN controller.

In the proposed managed WebRTC service, peers communicate through a WebRTC Signaling Server to establish a session by exchanging SDP and ICE (Internet Connectivity Establishment) objects. ICE objects contain IP address, transport protocol and port number of the peers. The signaling server also creates data sockets for each peer. The signaling server communicates with the SDN Controller via the *WebRTC Service Manager* (WSM) module using REST API. Besides passing peer information to the WSM, the signaling server also communicates back with the WebRTC peers to pass them the peer-to-peer available bandwidth and other session information received from the WSM.

WebRTC Service Manager (WSM): WSM module gathers session information about peers from the WebRTC signaling server through the REST API in order to create peer instances. The WSM organizes various services, such as dynamic routing, queue management, providing network statistics, and policing, that the NSP provides to RTC-VSP clients. WSM is also responsible to communicate the peer-to-peer available bandwidth information and other session information back to the WebRTC signaling server.

Traffic Engineering Manager (TEM): WSM module passes peer instance information to the TEM, which then (re)computes a path between peers and defines/updates flow tables at the switches accordingly. It also periodically allocates, updates, or destroys queues at the switches as explained in Section 2.4.3, and assigns WebRTC flows to the allocated queues. Finally, TEM passes per-flow bandwidth and other flow information back to the WSM module, which sends them to peers through the signaling server so that peers can select appropriate encoded video layers to be sent over these paths.

WebRTC Traffic Policing (WTP): WTP module is responsible to make sure that every WebRTC client is not exceeding the bandwidth that is reserved for them. If the

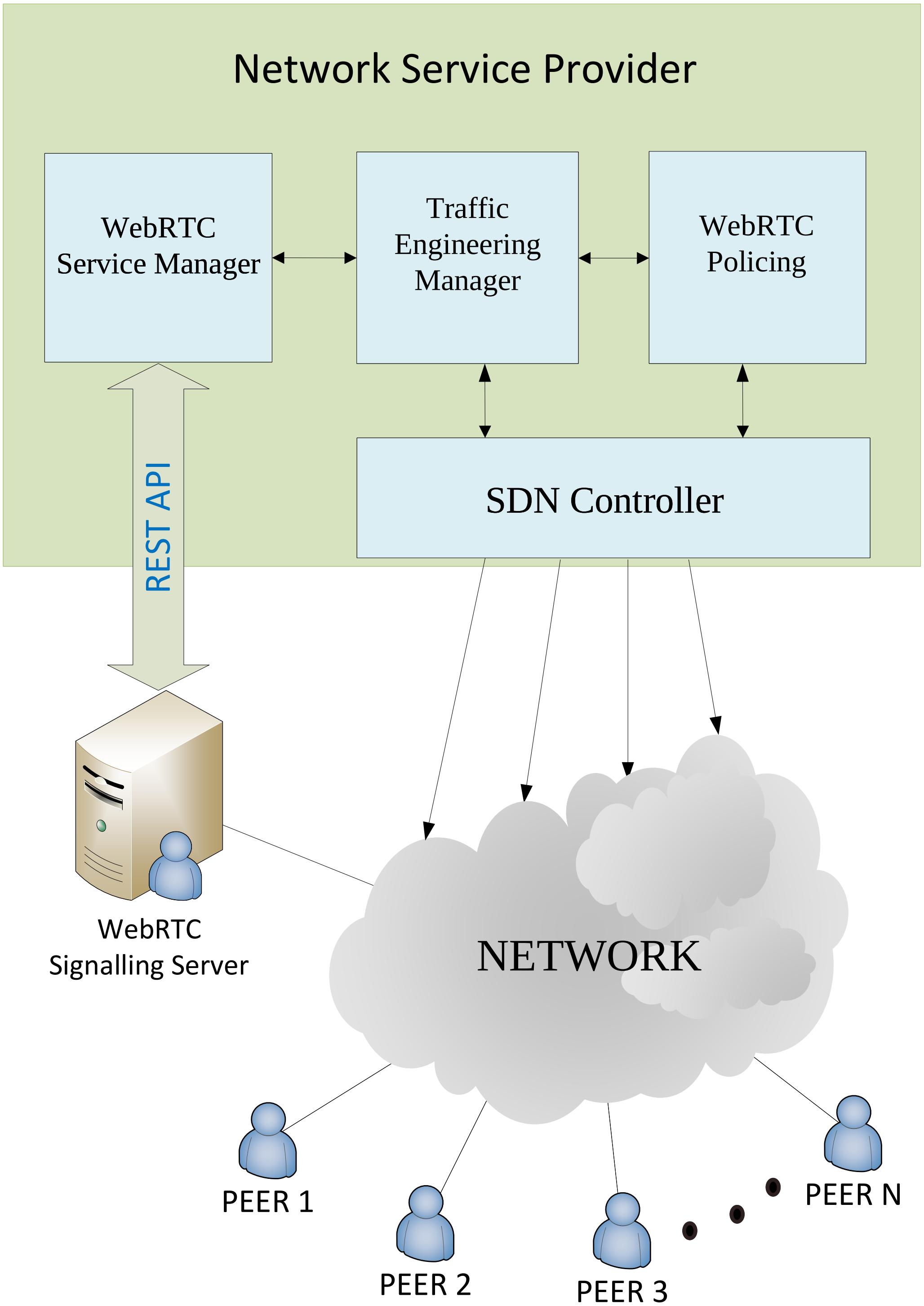

Figure 2.2: Architecture of managed WebRTC as value-added service over SDN.

client is exceeding that limit, WTP drops video packets to keep client's consumption less than or equal to the allocated bandwidth.

2.4.2 Constrained Shortest Path Algorithm

Best-effort WebRTC services use the shortest path algorithm to create media channel between peers. Since the SDN controller has full visibility of switch queue states and network traffic, the TEM can search for a path that satisfies some minimum bandwidth and maximum delay requirements, and the NSP can route managed clients over such paths. Hence, we propose the constrained shortest path (CSP) algorithm, summarized in Algorithm 1, for path computation between each pair of peers in both directions, since the path from peer1 to peer2 need not be the same as that from peer2 to peer1.

Algorithm 1: Constrained Shortest Path Algorithm

Input: G, src_i, dst_i, r_i
Output: p_i, bw_i
$bw_{low} = r_i.b_i^{min}$
$bw_{high} = r_i.b_i^{max}$
$bw_{mid} = \lfloor \frac{bw_{high} + bw_{low}}{2} \rfloor$
$p_i \leftarrow \emptyset$ (Set of links)
$bw_i \leftarrow$ Null
while $bw_{high} \geq bw_{low}$ **do**
    Create graph $G*$ such that $\forall$Link $\in G* \geq bw_{mid}$
    $p_i* \leftarrow$ Dijkstra($G*$, src_i, dst_i)
    $delay_{p*}$ = Sum of all latency delay over p_i*
    if $p_i* \neq \emptyset$ & $delay_{p*} \leq r_i.d_i^{max}$ **then**
        $p_i = p_i*$
        $bw_i \leftarrow$ min(bw over p_i*) (bottleneck)
        $bw_{low} = bw_{mid}$
        $bw_{mid} = \lfloor \frac{bw_{high} + bw_{low}}{2} \rfloor$
    else
        $bw_{high} = bw_{mid}$
        $bw_{mid} = \lfloor \frac{bw_{high} + bw_{low}}{2} \rfloor$

The inputs to Algorithm 1 are a directed network graph G that consists of vertices (V) and edges (E), and the service request r_i from the source peer src_i to destination peer dst_i, which contains the minimum bandwidth b_i^{min}, maximum bandwidth b_i^{max}, and the maximum delay d_i^{max}. The outputs are the optimal path p_i and the bandwidth bw_i that is allocated for client i over the path p_i.

The CSP algorithm performs a binary search between b_i^{min} and b_i^{max}. In each iteration, we create a graph by eliminating the links with bandwidth below bw_{mid}. Then, call the Dijkstra's algorithm and find the shortest path from src_i to dst_i on p_i*. If there exists a path such that the sum of all delays over all links in p_i* is below d_i^{max}, then it assigns this path to the current best path and searches for a path with a larger available bandwidth. Otherwise, it searches for a path with a lower bandwidth but with delay less than d_i^{max}.

Note that path computation is dynamic in the sense that it is repeated if the current path no longer satisfies the requirements in a dynamic network environment.

Complexity Analysis: Our search array consists of the set of consecutive integers from bw_{low} to bw_{high} Kbps with (bw_{high} - bw_{low}) elements. Since we are using binary search, in the worst case the while loop iterates for a maximum of $log_2(bw_{high}$ - $bw_{low})$ times. For each iteration, we create $G*$ with adjacency list data structure, which costs $O(|E| + |V|)$ and the Dijkstra's Shortest Path algorithm [15] with the advantageous structure of binary heap costs $O(|E|log|V|)$. The remaining operations in the loop cost $O(1)$. Therefore, the worst-case complexity of the proposed algorithm is $O\big(log\big(bw_{high} - bw_{low}\big)\ \big(|E| + |V| + |E|log|V|\big)\big)$.

2.4.3 Queue Management

End-to-end dynamic resource reservation is managed by directing managed WebRTC flows to priority queues that are set up on network switches by the SDN controller. There are two methods for dynamic queue management: In the first method, a different queue with the desired resources is set up for each new flow and destroyed when the WebRTC session ends. In the second method, a single priority queue, whose re-

sources are updated dynamically depending on the cumulative needs of all managed flows, is set up, and all managed flows are directed to this priority queue.

We have implemented the second method. In order to dynamically change the bandwidth of a queue according to accomodate , we periodically create a new queue with the new desired bandwidth, transfer all flows in the old queue to this new queue, and then destroy the old queue.

2.5 Evaluation

We first discuss the test environment in Section 2.5.1. Experimental results to evaluate the performance of the best-effort and managed WebRTC services are shown in Section 2.5.2.

2.5.1 Test Environment

We emulated a network with 20 virtual (OVS) switches using the Mininet [17] running on a server. The Floodlight SDN controller [16] runs as a separate application on the same server. The SDN controller is connected to the Mininet through a specified port and talks to the switches through its soutbound interface. The WebRTC signaling server runs as an application in one of the virtual hosts that is connected to the Mininet. We connected three real hosts (laptops) to the Mininet virtual switches using ethernet to USB hardware interfaces (hw-intf) in order to run WebRTC applications with VP9 encoders. The experimental set-up is depicted in Figure 2.3. We prefered to use real hosts to run the WebRTC application on separate hardware, since running the Mininet, SDN controller, WebRTC signaling server, and multiple instances of VP9 encoder/decoder on the same server can cause real-time CPU performance limitations.

2.5.2 Experimental Results

We compare the performances of our proposed motion-adaptive best-effort and managed value-added WebRTC services using scalable VP9 video coding with that of the default WebRTC service (provided by Google) using non-scalable VP9 video coding

using the evaluation environment with 3 mesh-connected real host peers as described in Section 2.3. We present the motion activity, video bitrate, spatio resolution, frame rate, the quantization parameter (qp), and PSNR values from peer1 to peer2 and peer3 in the presence of cross traffic for all service types. In the proposed best-effort motion-adaptive and managed WebRTC services, all clients perform scalable VP9 encoding at 400 kbps with 2 spatial layers where base layer is 320x240 and enhancement layer is 640x480, and 3 temporal layers for 30, 15 and 7.5 frame/sec, respectively. In the default (Google) system, non-scalable VP9 encoding is performed, where rate adaptation is performed by varying the qp between 2-63. In all scenarios, the average video bitrate for all service types are equal for a fair comparison. We recorded all videos as raw .yuv files for PSNR comparisons.

Streaming from Peer 1 to Peer 2: The Peer1 video has high motion between 20s-40s and low/no motion for the remaining times. The bandwidth between peer1 and peer2 is initially 400 kbps. For the best-effort service, cross traffic applied between

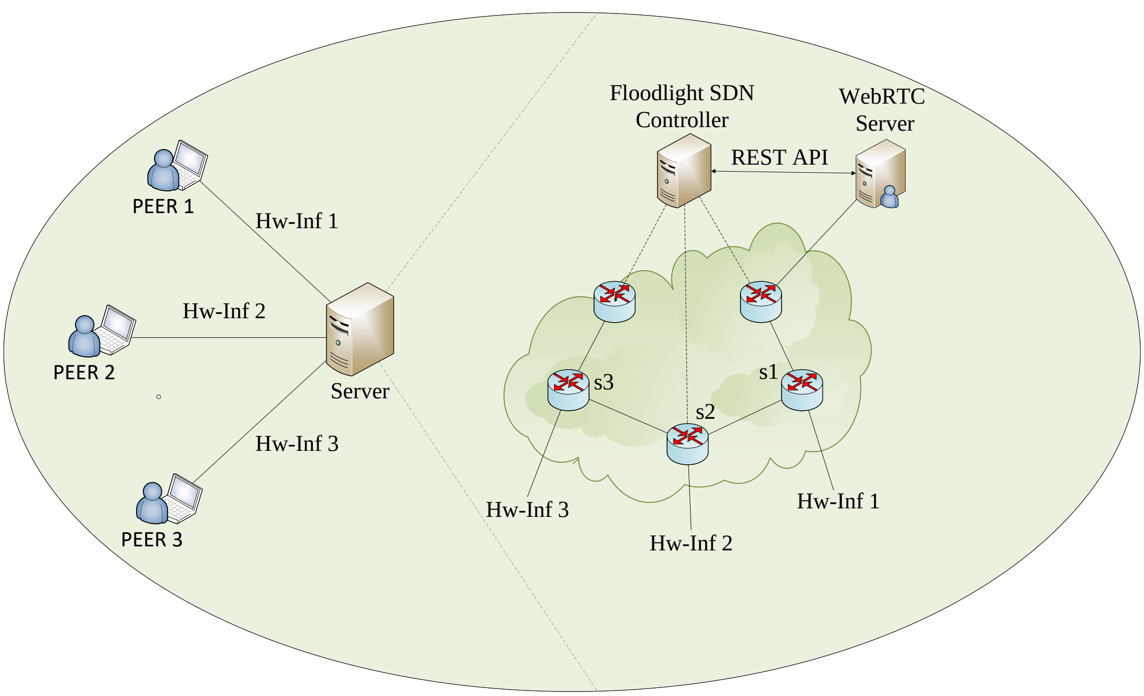

Figure 2.3: Test environment for mesh-connected multi-party WebRTC videoconferencing experiments over the Mininet.

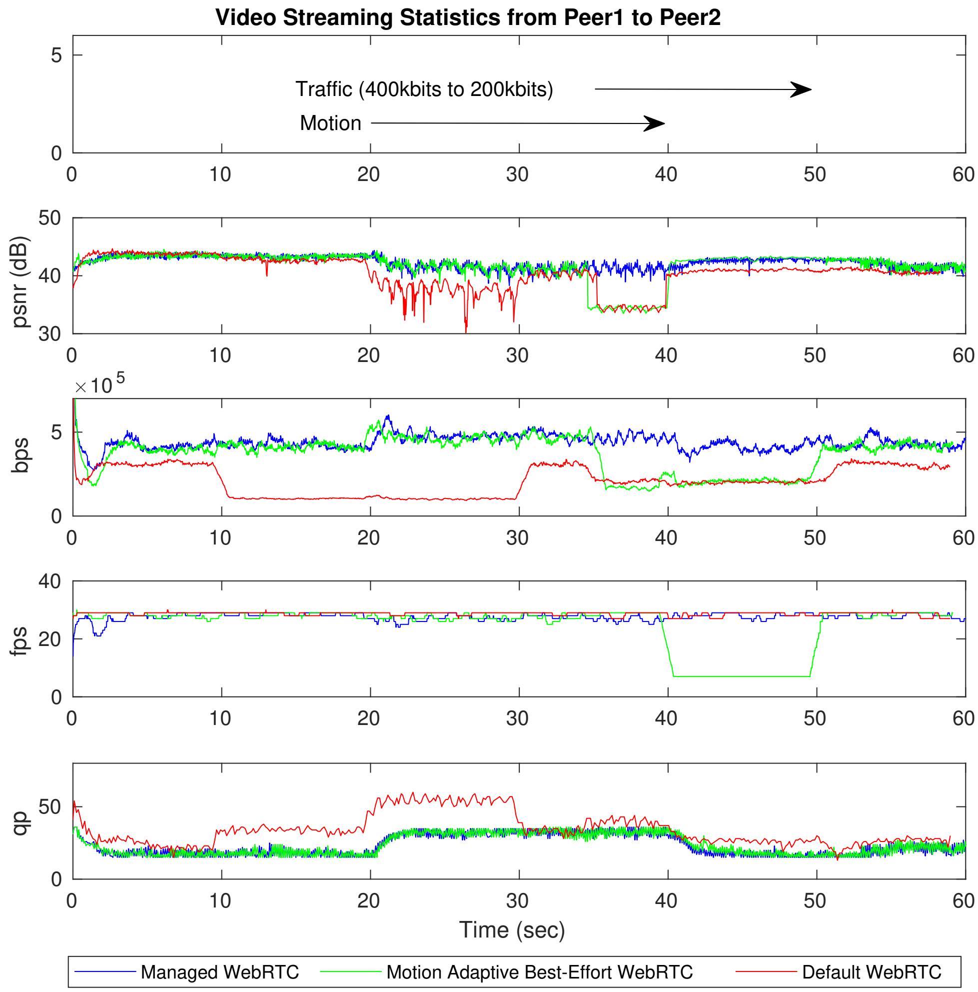

Figure 2.4: Results for video streaming from Peer 1 to Peer 2.

35s-50s reduces the available bandwidth to 200 kbps. The reserved bandwidth is not affected by the cross-traffic in the managed service. The motion-adaptive best-effort service drops a spatial layer in response to congestion between 35s-40s. Between 40s-50s when there is low motion, the best-effort services drops 2 temporal layers and retains both spatial layers to preserve video quality. The PSNR, frame rate, and qp values in this case are depicted in Figure 2.4.

Streaming from Peer 1 to Peer3: The available bandwidth is initially 300 kbps. Both the managed and best-effort services drop 1 temporal layer to adapt to 300 kbps.

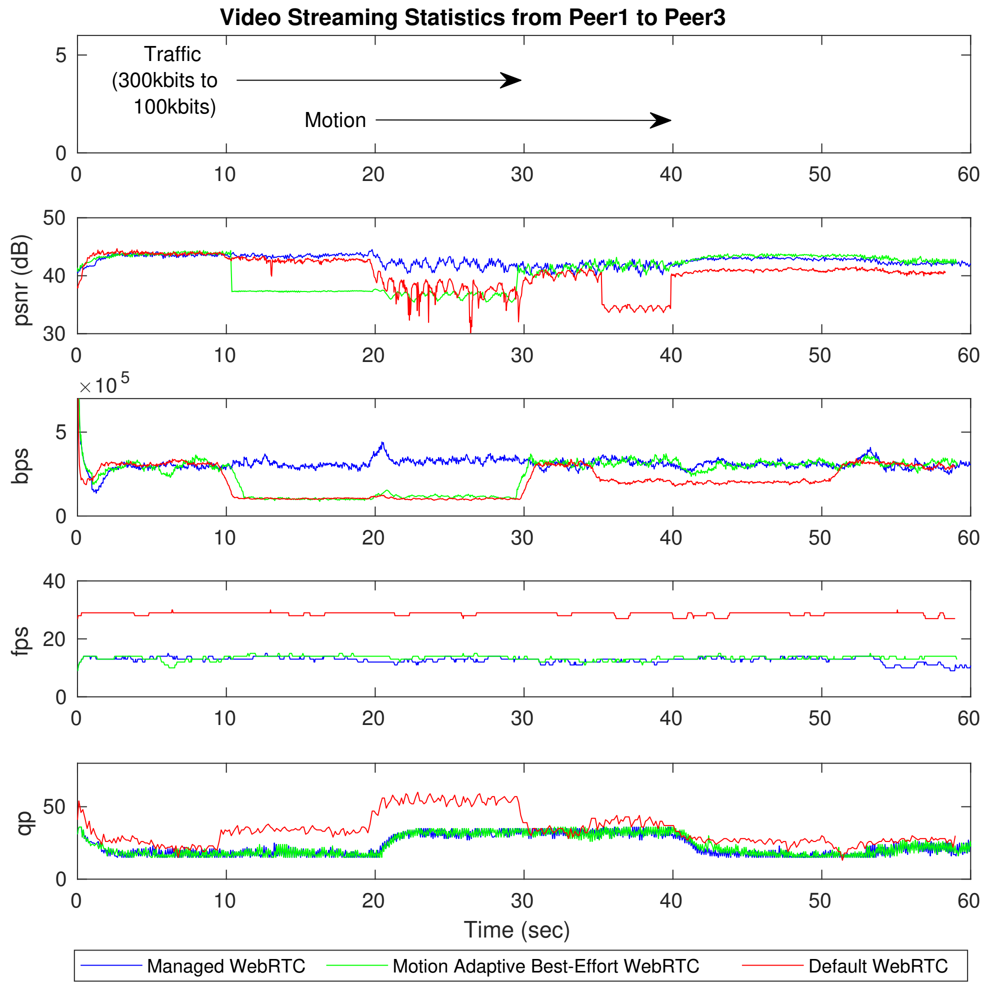

Figure 2.5: Results for video streaming from Peer 1 to Peer 3.

Between 10s-30s available bandwidth drops to 100 kbps and the adaptive best-effort service drops one spatial layer to preserve the frame rate. The PSNR, frame rate, and qp values in this case are depicted in Figure 2.5.

In all cases, we see that the managed service provides excellent and steady video quality. The motion-adaptive best-effort service adapts to congestion pretty well and provides acceptable visual video quality, while the default service either stalls or provides blurry pictures. Visual results of a picture during the congestion period are shown in Figure 2.6.

(a) (b) (c)

Figure 2.6: A frame from Peer1 to Peer 2: (a) Default service, (b) Motion-adaptive best-effort service, (c) Managed service.

2.6 CONCLUSIONS

Our main contributions are: i) we show that WebRTC community can make the transition from one-size-fits-all video coding to spatio-temporal scalable video coding with motion-adaptive rate adaptation for terminal resolution and network adaptive video transmission, and ii) WebRTC services can easily be offered as managed value-added services over SDN networks. Our results demonstrate that in both cases the video quality is superior to the default case of using non-scalable VP9 video coding both visually and in terms of PSNR.

Chapter 3

OPTIMIZATION OF SFU-CONNECTED WEBRTC VIDEOCONFERENCING SERVICES BY SDN-ASSISTED EDGE COMPUTING

3.1 Introduction

Recent advances in Software-Defined Networking (SDN), Network Function Virtualization (NFV), and data center infrastructure have enabled network service providers (NSP) to place network intelligence and services at network edges [8]. An example edge platform that is already being deployed by some NSP to support residential, mobile, and enterprise services is SDN-enabled broadband access (SEBA) [18]. Federated edge clouds supporting SDN-assisted network virtualization and slicing will play a key role in 5G networks to unlock the full potential of new and emerging services.

WebRTC [28] is a popular protocol for real-time communications that provides browser-to-browser voice, video, and data communications via simple Javascript APIs. VP9 [19] is a royalty-free codec that allows for efficient scalable video coding. WebRTC and VP9 are supported by leading browsers such as Firefox and Chrome. WebRTC with VP9 video encoding is currently available over *the best-effort* Internet. However, video quality often exhibit undesirable fluctuations due to sudden network bandwidth variations even though WebRTC employs powerful network estimation and rate adaptive video encoding schemes. Hence, it is particularly important to manage real-time video communications services at network edges to provide low latency and guaranteed bandwidth.

The objective of this paper is to propose an edge-cloud integrated service architecture to offer low-delay, guaranteed quality premium WebRTC service with the

following features:

- guaranteed bandwidth to all parties in a multi-party WebRTC session within edge access networks
- minimum end-to-end delay between all pairs of parties in a multi-party WebRTC session
- minimum overall network bandwidth consumption.

While the first two features benefit end user, the last one benefits the NSP. The guaranteed bandwidth within edge networks provides each party predictable and consistent video quality throughout a session. We assume that bandwidth is largely overprovisioned in the core network; hence, no bandwidth reservation is needed within the core network.

The rest of this paper is organized as follows: Related work and novelty of the paper are discussed in Section 3.2. Section 3.3 presents the NSP network topology including edge clouds. The proposed edge-integrated premium WebRTC service architecture is described in Section 3.4. Service emulation and evaluation are discussed in Section 3.5. Finally, Section 3.6 presents conclusions.

3.2 Related Work and Novelty

A proprietary multi-party videoconferencing system using scalable video coding was first proposed in [9] in order to avoid transcoding at the multi-point control unit (MCU). Scalable coding is now a mainstream technology in the best-effort multi-party videoconferencing together with a central video router, called selective forwarding unit (SFU), that selectively forwards layer streams received from sending clients according to terminal type and connection bandwidth of receiving clients.

Alternatives to the central SFU multi-party service architecture are the peer-to-peer (P2P)-MCU architecture, where the MCU functionality is hosted in one of the peers [11], and implementation of the MCU or SFU in the cloud [12].

Prior work on managed WebRTC videoconferencing services over SDN is very limited. Launching WebRTC services in SDN environment was discussed in a concept paper on "network as a service" [13], which did not present an implementation. Realizing dynamic-network-enabled RTC on a proof-of-concept 5G network was discussed in [14]. Our previous work on managed multi-party WebRTC services considers the case of mesh-connected clients over SDN [1] without considering SFU-connected service or edge cloud concepts. However, the upload bandwidth of peers is a bottleneck in multi-party WebRTC with mesh topology.

The novelties of this paper are:

- managing WebRTC at network edges to provide guaranteed bandwidth at edge access networks
- optimization of SFU service to minimize end-to-end delay and overall network resource usage
- a distributed SFU framework, which decreases delay and avoids single point of failure.

To the best of our knowledge, there is no prior work on optimization of WebRTC service using SFU at network edges to directly compare with our architecture or methodology.

3.3 Network Service Provider Topology

This section briefly summarizes the NSP topology. Section 3.3.1 introduces the edge cloud architecture that controls the access network domain and hosts services such as the WebRTC service proposed in Section 3.4. Dynamic resource reservation at the network edges is discussed in Section 3.3.2.

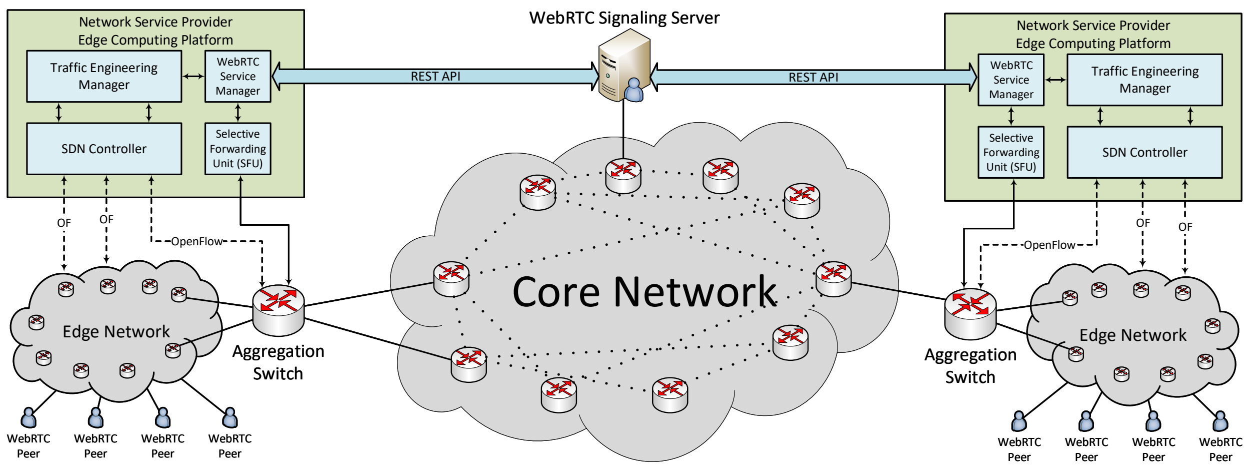

Figure 3.1: NSP topology, including the core network and two edge clouds, and the edge-managed WebRTC service architecture.

3.3.1 Edge Clouds

The NSP network consists of a core network and a number of access network domains controlled by edge clouds as depicted in Fig. 3.1. Each edge cloud supports SDN/NFV infrastructure to control an access network domain serving residential and mobile broadband customers. We assume each edge cloud has an aggregation switch that is the gateway between local switches within the domain and the core network. Each edge cloud also hosts services such as the WebRTC service proposed in this paper.

SDN Controller (SDN-C): Each edge network is controlled by SDN. Hence, each edge cloud has a compute node running an SDN controller (SDN-C). SDN-C communicates with switches using OpenFlow through its south-bound interface, and with applications such as the traffic engineering manager (TEM) and WebRTC service manager (WSM) using the REST API through its north-bound interface.

WebRTC Service Manager (WSM): WSM module gathers session information about peers from the WebRTC signaling server through the REST API in order to create peer instances. The WSM manages various services that the NSP provides to WebRTC peers, such as resource reservation, queue management, providing network statistics, and policing. WSM is also responsible to communicate resource

reservations and other session information back to the WebRTC signaling server.

Traffic Engineering Manager (TEM): WSM module passes peer instance information to the TEM, which then (re)computes a path between peers and defines/updates flow tables at the switches accordingly. It also periodically allocates, updates, or destroys queues at the switches, and assigns WebRTC flows to the allocated queues. Details of these steps are discussed in Section 3.3.2. Finally, TEM passes per-flow path reservation and other flow information back to the WSM module, which sends them to peers through the signaling server so that peers can select appropriate encoded video layers to be sent over these paths.

3.3.2 Dynamic Resource Reservation at Edges

The TEM computes bandwidth and delay constrained paths between end-points and oversees implementation of these paths at SDN switches by queue management as follows:

Constrained Shortest-Delay Path Finder Algorithm

Best-effort WebRTC services use the shortest path routing to create media channel between peers. Since the SDN controller has full visibility of switch queue states and network traffic in each edge network, the TEM can search for a path that satisfies desired bandwidth requirements, and the NSP can route premium WebRTC peers over reserved paths. We employ the bandwidth-constrained shortest-delay path finder (CSPF) algorithm, summarized in Algorithm 2, for path computation between endpoints in both directions, since the uplink and downlink paths between endpoints need not be the same.

The inputs to Algorithm 1 are a directed network graph G that consists of vertices (V) and edges (E), and the desired bandwidth bw_i from the source peer $sender_i$ to destination peer $receiver_i$. The outputs are the optimal path p_i^* and the minimum delay $delay_i$ over the path p_i^* with hop count hc_i.

The CSPF algorithm performs a sub graph creation G^*, trimming the links with

Algorithm 2: Constrained Shortest-Delay Path Finder

Input: G, $sender_i$, $receiver_i$, bw_i
Output: p_i^*, $delay_i$, hc_i
$p_i^* \leftarrow \emptyset$ (Set of links)
$delay_{p^*} \leftarrow$ Null
$hc_i = 0$
Create graph G^* such that $\forall$Link $\in G^* \geq bw_i$
$p_i^* \leftarrow$ Dijkstra(G^*, src_i, dst_i)
$delay_{p^*}$ = Sum of all latency delay over p_i^*
hc_i = Number of links over p_i^*

available bandwidth below the desired bandwidth, so that each link in G^* has at least bandwidth bw_i available. Then, it runs Dijkstra's Shortest Path algorithm [15], where link delays are considered as cost and finds the path that satisfies required bandwidth value bw_i with the minimum delay $delay_{p^*}$.

Complexity Analysis

The complexity of creating the graph G^* with adjacency list data structure is $O(|E|+|V|)$ and running the Dijkstra's Shortest Path algorithm [15] with binary heap is $O(|E|log|V|)$. Therefore, the worst-case complexity of Algorithm 2 is $O\big(|V|+|E|log|V|\big)$.

3.4 Edge-Based WebRTC Service Architectures

This section introduces a framework for multi-party WebRTC videoconferencing managed at network edges as a value added service with assured quality of service that can be provided by the NSP at network edges. The overall system architecture including collaboration between NSP edge clouds and the WebRTC service provider (WSP) is presented in Section 3.4.1. In a standard multi-party WebRTC service with scalable video coding and a single selective forwarding unit (SFU), the location of the SFU in the network plays an important role in the end-to-end delay performance. The options for placing the SFU are discussed in Section 3.4.2. Section 3.4.3 proposes a novel multiple (distributed) SFU framework for our edge-managed premium

WebRTC service architecture, where an SFU is placed in every edge cloud, and a single WebRTC session may use more than one SFU. We discuss the implementation of resource reservation in the edge networks in Section 3.4.4.

3.4.1 Managed Service and NSP-VSP System Architecture

The main design principle of WebRTC is to generate matching PeerConnection (PC) object pairs to send and receive data for point-to-point communication between pairs of endpoints, i.e., each receive PC object is connected to only one send PC object. PC objects are the main elements in order to implement different streaming architectures. In a mesh-connected N-party WebRTC session each peer creates $N-1$ send PC and $N-1$ receive PC objects. As a result, in a mesh-connected session, the upload bandwidth of peers forms a bottleneck. In an SFU-connected session, each peer creates just one send PC (to the SFU) and $N-1$ receive PC objects. Hence, the upload bandwidth bottleneck is avoided. An SFU implementation requires generating PC objects that receives stream from a peer, duplicates it and forwards to each receiving peer through a different PC object. Whether it is mesh or SFU connected media streaming, all peers must first connect to the signaling server to establish a WebRTC session.

WebRTC Signaling Server: Peers establish a session by exchanging Session Description Protocol (SDP) [21] and Internet Connectivity Establishment (ICE) [22] objects through a WebRTC Signaling Server. ICE objects contain IP address, transport protocol and port number of the peers. The signaling server also creates data sockets for each peer. There exists two-way communication between the WebRTC signaling server and the SDN Controller via the WSM module of the NSP. Besides passing peer information to the WSM, the signaling server receives path reservation confirmation from the WSM and passes them back to the peers.

A traditional WebRTC session does not collaborate with the NSP; thus, it is a best effort service with no control over network resources. In the proposed edge-managed WebRTC service, the WSP collaborates with the NSP, and the NSP offers per-flow

quality of service (QoS) by computing minimum delay paths with specified bandwidth within edge networks as described in Section 3.4.4, and implements path reservations by queue management at switches.

The WebRTC service orchestration between the peers, SFU(s) and edge network controllers is realized by the *Signaling Server* (SS). The SS sends session information to all peers, SFU servers and WSM modules of different edge domains. SS keeps a dictionary with peer IP-addresses and their matching edge network SFU IP-addresses as key-value pairs so that SS knows which SFU server each peer is connected to over the entire topology.

3.4.2 Single SFU Service

In single SFU architecture, every client in a session sends and receives a video stream from a single central SFU. Suppose N peers join a session. Then, each peer creates one send-only PC object (sPC) and $N-1$ receive-only PC object (rPC). The SFU, on the other hand, creates N rPC objects and $N(N-1)$ sPC objects (N^2 PC objects in total).

The single SFU service architecture is depicted in Figure 3.2, where there are 3 peers in a session and the video stream originating from each peer is shown with a different color. There are two options, depending on the location of the SFU:

Random Anywhere-SFU Service

This is the most common implementation of SFU streaming service architecture. An SFU server is randomly placed in the global network that consists of all the edge networks and the core network.

Random Edge-SFU Service

An SFU server is placed in each edge network next to the aggregation switch as depicted in Figure 3.1. All peers in the same session connect to a single SFU server located in the edge network of one of the peers (randomly selected) in that session.

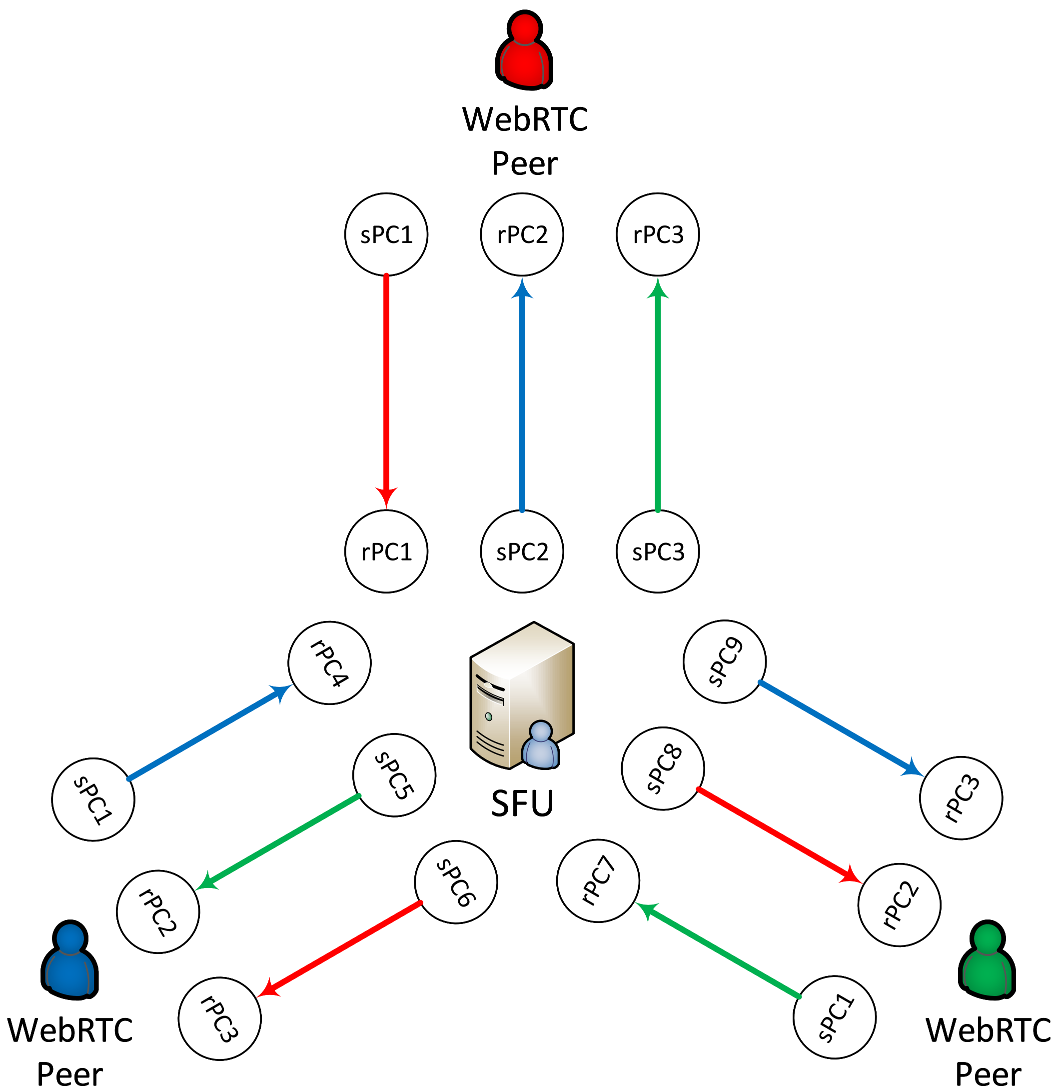

Figure 3.2: Single SFU service logical system architecture.

3.4.3 Distributed Edge-SFU Service

In the proposed multiple SFU architecture, an SFU is placed in each edge network and a session can be served by more than one SFU, where each peer connects to the SFU located in its own edge network. Assume that there are N peers in a session. The number of SFU that is going to be used for this session can be between 1 and N depending on whether all peers are located in different edge networks or not.

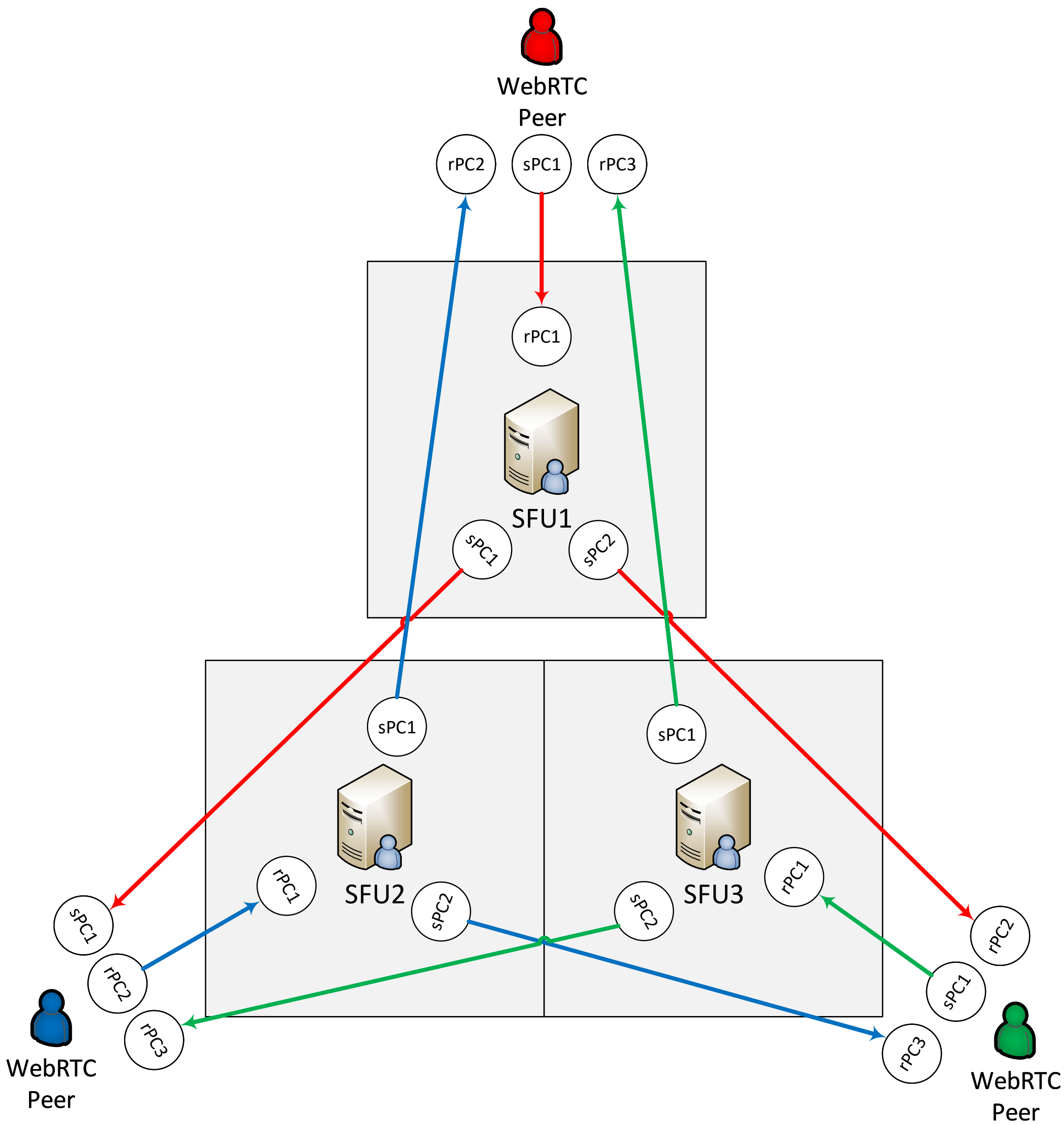

Figure 3.3: Multiple SFU logical system architecture.

The proposed multiple SFU service architecture is depicted in Figure 3.3, where there are 3 peers in a session and 3 SFUs are used because all peers are in different edge networks. Video stream from each peer is shown by a different color. Each peer connects to the SFU that resides in its own edge network. In the most general case, some peers can be in the same edge network, and connect to the same SFU server.

The total number of PC objects created in SFU servers for a single session is still N^2.

A WebRTC session with the distributed SFU architecture can be instantiated by pairing receive-only PC objects of each peer to send-only PC objects of SFUs located at different edge clouds and vice versa. Experimental results show that this proposed SFU service architecture is superior to random anywhere-SFU and random edge-SFU architectures in terms of both E2E delay and NSP resource management.

3.4.4 Resource Management in Edge Networks

In the SDN-enabled broadband access architecture, each edge network has its own SDN controller that is responsible for network control. WebRTC signaling server (SS) talks to *WebRTC Service Manager* (WSM) at each edge via the REST API to pass SDP and ICE objects of peers in that edge network and SDP and ICE objects of SFUs at other edges. WSM classifies all flows to be set up as internal or external flows by analyzing their origin and destination IP addresses and passes resource reservation requests to the *Traffic Engineering Manager* (TEM), which is a controller application. Since the underlying topology of the edge network is fully visible to the TEM, the TEM reserves path and bandwidth for both send and receive video streams in a session depending on whether they are internal or external flow as follows:

Internal Flows

When the $sender_i$ and the $receiver_i$ are in the same edge network, TEM runs Algorithm 2 to find a bandwidth constrained, minimum delay path between these end points. Then, the SDN controller allocates priority queues matching the flow along the computed path.

External Flows

TEM runs Algorithm 2 between the aggregation switch and the end-point that can be either SFU or peer. Then, at each edge, the related SDN controller allocates priority queues matching the flow along the computed path.

3.5 Evaluation

We first describe a small scale emulation of the proposed edge-managed WebRTC service architecture in Section 3.5.1 as a proof of concept with real network protocols and service elements. Simulation results to evaluate the performance of proposed service architectures on a large scale network are presented in Section 3.5.2.

3.5.1 Emulation Environment

We emulated an NSP with a core network and three edge domains by running Mininet [17] instances on four separate laptops. Mininet instances emulating edge domains have 10 open virtual switches (OVS) and the core network has 30 OVS. In every edge domain, one switch has been designated as aggregation switch that is connected to one of the switches in the core network using GRE tunnels. Each edge domain is controlled by a separate instance of the Floodlight controller [16], which communicates with the switches in its domain through its southbound OpenFlow interface. We also set up a Janus WebRTC server [23] in each edge domain as a virtual host to act as an SFU with a connection to the aggregation switch. Each edge domain also has a virtual host that is a WebRTC peer browser connected to a randomly selected switch. We use multiple laptops in our emulation, because running the Mininet, SDN controllers, Janus WebRTC server, and instances of VP9 encoder/decoder on the same server causes real-time CPU performance limitations.

The *Signaling Server* runs as a virtual host that is connected to a randomly selected core switch. *Signaling Server* communicates with SDN controllers at each edge domain as explained in Section 3.4.4. Our emulation results show that the proposed service architecture successfully runs over real network and WebRTC protocols using Floodlight as a proof of concept.

3.5.2 Simulation Results

Since it is difficult to build a large scale emulation environment, we provide performance evaluation results over a simulation environment using a core network with

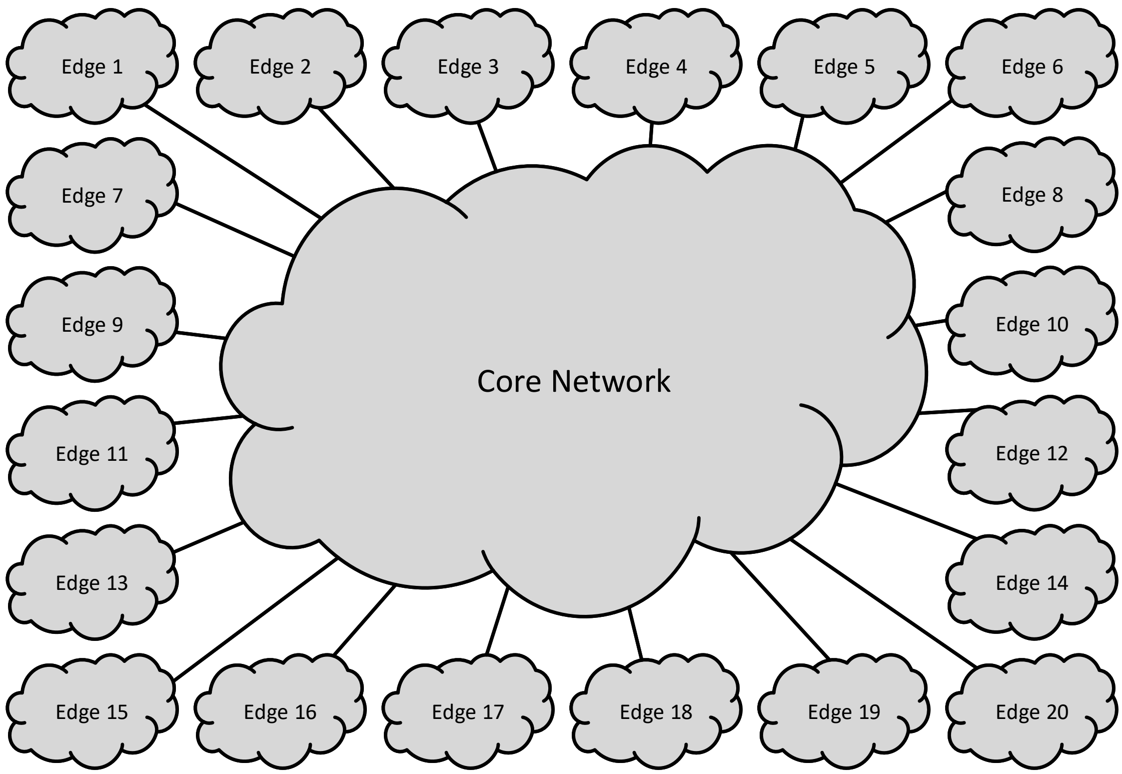

Figure 3.4: Edge-Core Network Architecture with 20 Edge Domains.

thousands of switches and twenty edge domains as depicted in Figure 3.4 and one hundred WebRTC sessions involving hundreds of peers. Each edge network is connected to the core network with a single aggregation switch.

NSP Simulation

We simulated core and edge networks of different sizes with random topology, where each edge network has 16, 25, 36, 49, 64, 81, 100 switches and core network has 256, 400, 576, 784, 1024, 1296, 1600 switches, respectively. We employ a physical distance based probabilistic model [24] to create bi-directional links between pairs of switches in both core and edge networks. In order to model physical distances between nodes, we consider a rectangular grid of unit squares, where there are as many squares as the number of switches in both core and edge networks. Each switch is placed at a

random coordinate within a unit square. The probability P_{ij} of a link between two nodes i and j is given by

$$P_{ij} = \beta \exp\left(\frac{-d_{ij}}{L\alpha}\right)$$

where d_{ij} is the Euclidean distance between the nodes, L is the maximum Euclidean distance between any two nodes, and α and β are parameters. In the simulations, we assume that each link introduces a fix delay of 5 msec. This value is determined as the average of *Traceroute* measurements [25] from our server to different university servers across the globe.

WebRTC Service Simulation

We compare performances of edge-managed WebRTC services with random edge-SFU and distributed edge-SFU with that of the random anywhere-SFU, which is not an edge based solution, in terms of i) mean of maximum E2E delay of all WebRTC sessions, which is a service quality measure relevant to end users, and ii) the total bandwidth consumption of WebRTC services over the edge and core network, which is a measure relevant to the NSP. Note that the video quality in all cases will be the same, since we assume the same bandwidth is reserved for all peers in all cases within edge networks.

In the proposed managed WebRTC service, each peer runs a scalable video encoder configured to encode video with two spatial layers, 360p base layer at 350 kbps and 720p enhancement layer at a total bitrate of 1 Mbps, so that peers receive guaranteed quality video at a pre-specified rate according to their terminal type. All peers with terminal type HD receive both layers and peers with SD terminal receive only one layer video. The VSP passes the terminal types of clients to the TEM unit of NSP, and TEM reserves bandwidth between end points within its edge cloud.

We picked the proportion of device types (received video resolution) for each peer according to the numbers projected for 2021 in the report [26]. The estimated distri-

Table 3.1: PSNR(dB) of scalable coded videos according to spatial layer.

Layer	FP	Johnny	K&A	Vidyo3	Vidyo4
720p	37.59	38.82	38.08	36.45	37.13
360p	33.30	35.92	33.37	31.61	33.19

bution of video terminal resolutions are 41%, 47% and 12% for SD, HD, and UHD, respectively. Since we only considered 720p and 360p video in our experiments, we set the probability that a peer has 360p terminal as 0.4 and 720p terminal as 0.6.

Test Videos: We used five 720p raw test videos (with .yuv extension), namely FourPeople (FP), Johnny, KristenAndSara (K&A), Vidyo3, Vidyo4 that are available in [27]. There is no global camera motion or large motion content in these videos, which is typical of a real-time videoconferencing session. We encoded these videos using scalable VP9 with two spatial layers, where base layer is 360p encoded at 350 kbps and enhancement layer is 720p encoded at 1 Mbps (including the base layer). The PSNR values of the encoded videos at these bitrates are shown in Table 3.1 that indicate good quality. Since videos are delivered over paths with reserved bandwidth, the received videos have the same PSNR provided in Table 3.1.

Service Delay and Bandwidth Usage Results

For each network size, we ran tests for random-anywhere, random-edge, and distributed-SFU scenarios, with 100 WebRTC sessions each having 3, 4, and 5 peers, respectively. We repeat each experiment 100 times with random switch positions within their respective squares and randomly created links in edge and core networks. The peers are connected to randomly chosen switches in randomly selected edge networks. In the random anywhere-SFU architecture, all peers sign in to an SFU that is connected to a randomly selected switch anywhere in the core or edge networks. In the random edge-SFU architecture, all peers sign in to an SFU located in the edge cloud of one of the peers in the session. Unlike the current practice of using a single SFU for a session, the distributed edge-SFU uses more than one SFU for a session. Each peer

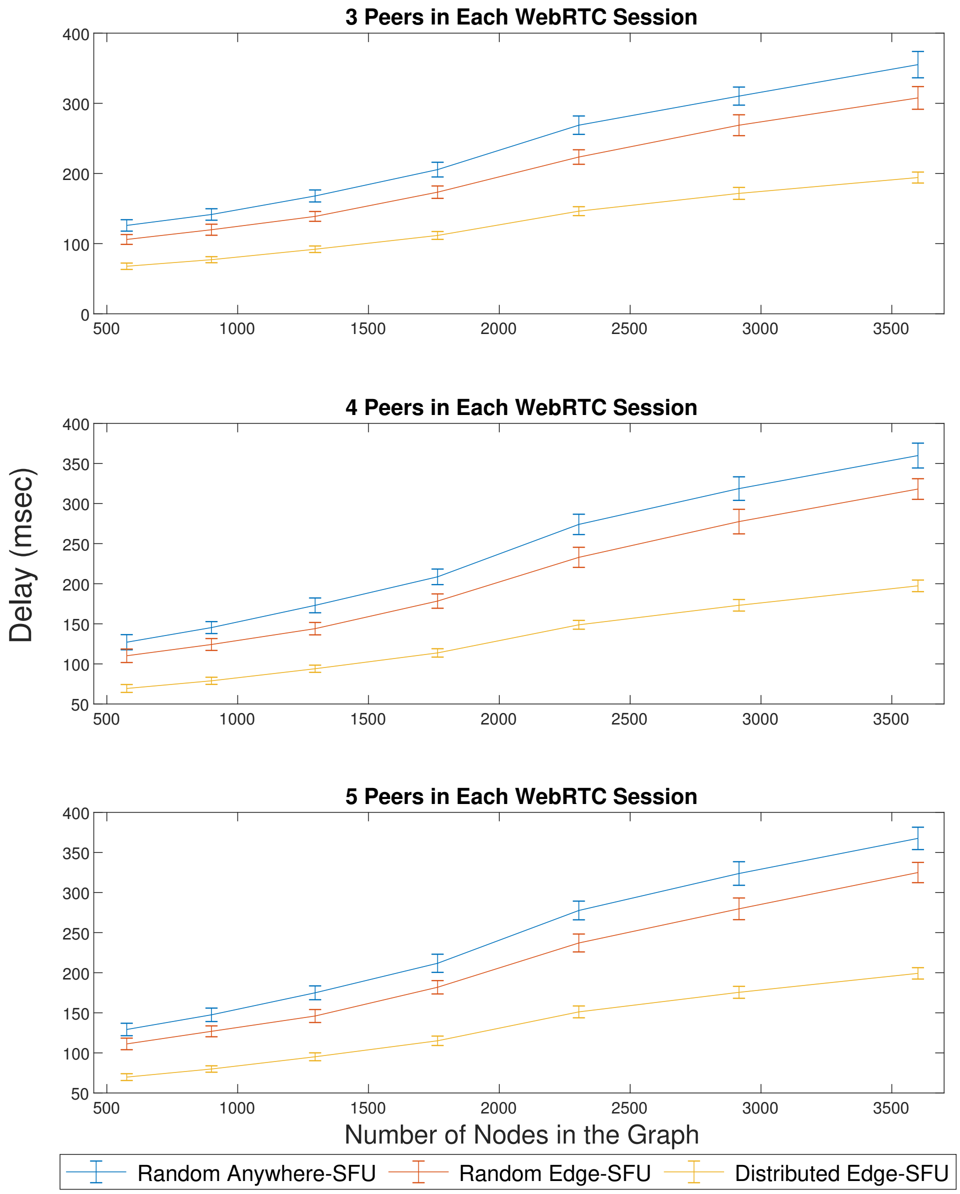

Figure 3.5: Mean of the maximum E2E delay in 100 trials of 100 services.

is connected to the SFU in its own edge cloud with its sender PeerConnection object (sPC) and other peers connect to this SFU with their receiver PeerConnection (rPC) objects as explained in Section 3.4.3. As a special case, if all peers are in the same edge network, there will be single SFU for that session.

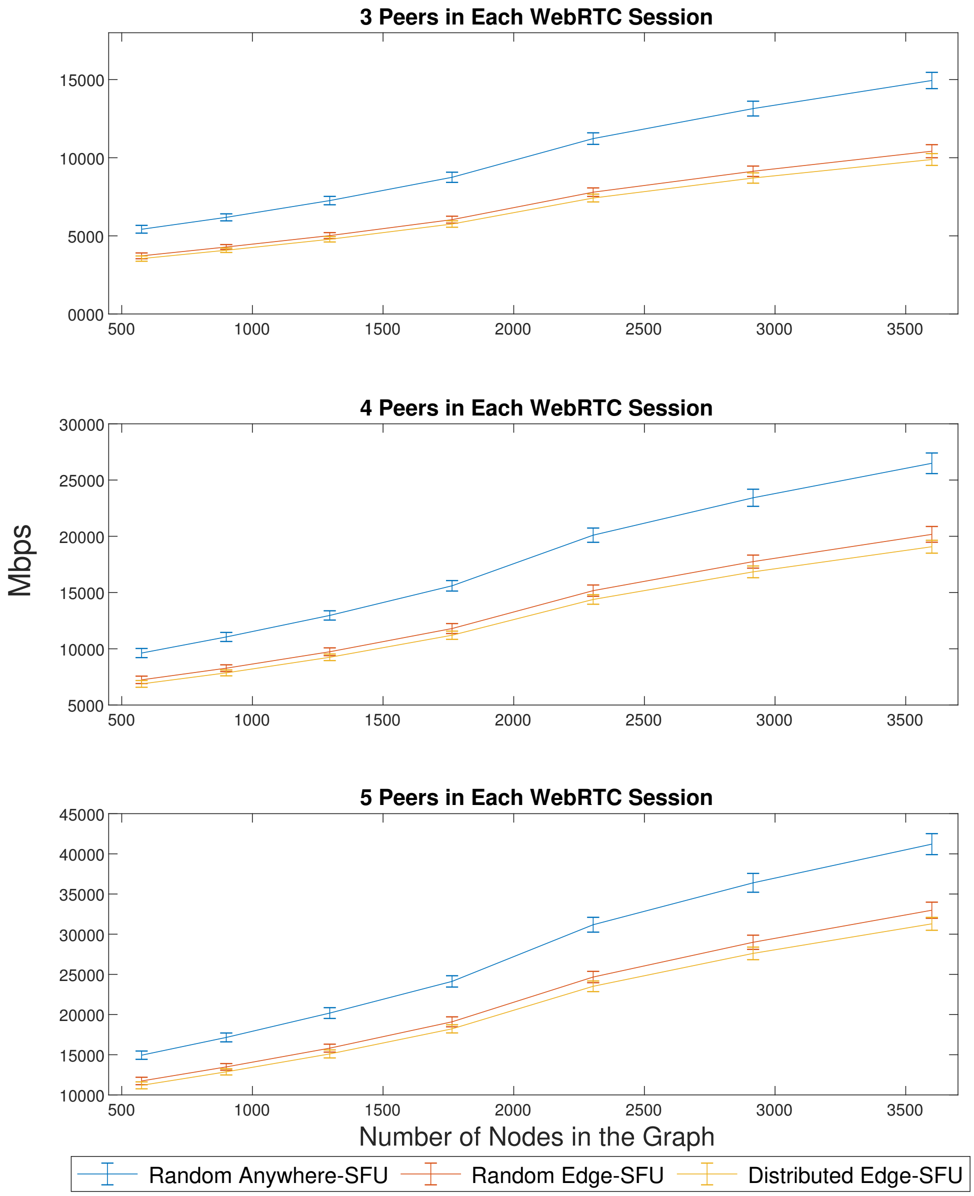

Figure 3.6: Mean of E2E bandwidth consumption in 100 trials of 100 services.

The average and standard deviation of maximum delay and total bandwidth among all the 100 sessions with 3, 4, and 5 peers per session vs. network size are shown in Fig. 3.5 and Fig. 3.6, respectively. Delay results show the average over 100 trials of the maximum E2E delay among the 100 WebRTC sessions in each trial.

Results show that, first of all, SFU deployment in edge networks significantly reduces network resource consumption and total video delay. Furthermore, delay and bandwidth results reveals that the proposed distributed WebRTC service offers value to both NSPs and end users. This is because the distributed SFU architecture consumes less overall network resources and it provides much better worst case E2E delay performance, compared to the random, random-anywhere and random-edge SFU architectures. These observations are consistent over all network sizes tested.

3.6 Conclusions

We propose an edge-cloud integrated managed multi-party WebRTC service architecture for NSPs or NSP-authorized third-party service providers to provide such services over reserved network slices. The proposed architecture includes a modified signaling server and a NSP-WebRTC service manager that orchestrate the collaboration between the NSP and the WebRTC service. We implemented an emulation environment using the Mininet with real OVS and SDN controllers for each edge network to validate the proposed architecture.

Our simulation results clearly demonstrate that i) in case all peers connect to a single SFU, placing the SFU at a network edge, where at least one of the peers reside lowers the mean of maximum delay compared to placing the SFU anywhere in the core network or edges, ii) using multiple SFU servers, where each peer connects to the SFU in its own edge domain to send its outbound stream provides an even lower end-to-end delay. Moreover, using multiple SFUs also results in the lowest overall total use of network resources for each WebRTC session. Hence, the proposed multiple SFU service is beneficial for both end users (lower delay) and for the NSP (lower use of network resources) compared to today's single-SFU WebRTC communications. We note that there is no difference in received video quality for these different SFU configurations, since we allocate the same reserved (guaranteed) bandwidth in the edge networks in all cases.

Chapter 4

WEBRTC VIDEOCONFERENCING SERVICES USING SDN-ASSISTED IP MULTICASTING AND SVC

4.1 Introduction

WebRTC is a popular protocol for real-time browser-to-browser voice, video, and data communications over the Internet using simple JavaScript APIs. VP9 is a royalty-free video codec that allows for efficient scalable video coding. We review them in Section 1.2. They are both supported by the leading browsers such as Firefox and Chrome.

The current service model to offer multi-party WebRTC videoconferencing between peers with heterogenous network connection and terminals over the best-effort Internet requires deploying a central server/router, called the selective forwarding unit (SFU), where each peer sends a scalable encoded video stream to the SFU. This model avoids the upload bandwidth bottleneck associated with mesh connection model; however, it increases peer delay and overall network load compared to mesh model in addition to requiring investment in high speed SFU routers with high bandwidth connection. Note that predictable bandwidth and stringent low-delay requirements of real-time communication cannot always be met by cloud processing and over-the-top (OTT) best-effort services.

The emergence of software-defined networking (SDN) and network function virtualization has enabled network service providers (NSP) to implement multi-tenant network slicing and IP multicast easily. Hence, provisioning video services with reserved bandwidth and guaranteed delay managed by the NSP is a subject of high interest in the upcoming 5G networks. This paper introduces a new managed multi-

party WebRTC service model using scalable video coding based on: i) innovative use of SDN-assisted Internet Protocol (IP)-multicasting that is compliant with the WebRTC protocol, and ii) use of NSP-managed network slices to deliver WebRTC content with predictable bandwidth and delay. The proposed SDN-assisted IP multicasting WebRTC service performs selective duplication and forwarding of scalable video layers at any SDN switch in the network under the supervision of the SDN controller, while the SFU model does so at a special central server/router; hence, the proposed method can be considered as a distributed implementation of the SFU at optimally selected SDN switches across the network.

The paper is organized as follows: Related works and novelty are discussed in Section 4.2. Section 4.3 introduces the proposed WebRTC service model with SDN-assisted IP multicast using scalable video coding. Experimental results are provided in Section 4.4. Section 4.5 concludes the paper.

4.2 Related Works and Novelty

We review prior art on use of scalable video coding in videoconferencing, SDN-assisted IP multicasting, and WebRTC services over SDN and then discuss the novelty of this paper.

A proprietary multi-party videoconferencing system over the Internet using scalable video coding was first proposed in [9] in order to avoid transcoding at the multipoint control unit (MCU). This is now a mainstream solution in multi-party videoconferencing and the term SFU was coined for the central router that selectively forwards video layers to peers according to their requirements. Other recent solutions for multi-party WebRTC videoconferencing include P2P-MCU architecture, where MCU functionality is hosted in one of the peers [11] and implementation of the MCU or SFU in the cloud [12].

Current IP multicast routing standards employ the shortest-path trees for distributed multicast routing. A receiver-driven layered IP multicast protocol, which combines scalable video coding and IP multicast groups subscribed by clients, was

first proposed by McCanne et al. [32]. This was later extended to a hybrid sender- and receiver-driven approach [33]. In a centralized SDN setting, one can construct the Steiner tree to find a multicast tree with the minimum cost. Since computation of the Steiner tree is an NP-hard problem, approximate methods for SDN-assisted uni-directional IP multicast has been proposed [34–37]. Sheu et al. [34] present two algorithms: a minimal path cost algorithm to minimize communication cost from the source to clients under delay constraint and a shortest path maximum bandwidth utilization algorithm to maximize overall link utilization. Shen et al. [35] propose a new reliable multicast tree assisted by SDN, called recover-aware Steiner tree (RST). However, to the best of knowledge there are no works that address SDN-assisted lowest-delay bi-directional IP multicasting within the WebRTC framework.

Prior work on managed multi-party WebRTC services over SDN is very limited. Launching WebRTC services in SDN environment was discussed in a concept paper on ”network as a service” [13], which did not have an implementation. Implementation of dynamic-network-enabled RTC on a proof-of-concept 5G network was discussed in [14].

In our previous work, we proposed a managed WebRTC service architecture for mesh connected clients [1]. However, use of a central SFU in an end-to-end managed WebRTC service or distributed implementation of SFU functionality at network switches are not discussed in the literature before.

The main novelties of this paper include: i) We develop a novel managed WebRTC service architecture using WebRTC-compliant SDN-assisted IP multicasting of scalable video for a distributed implementation of SFU functionality at network switches. ii) We propose a new algorithm for construction of delay-aware IP-multicast tree as an SDN controller application given the desired bitrate for peers. The proposed service architecture provides lower end-to-end delay and overall network resource usage compared to the central SFU service model.

4.3 NSP-Managed Multi-party WebRTC Service using SDN-Assisted IP Multicasting and SVC

This section introduces the main contribution of the paper. We first provide an overview of the proposed multi-party WebRTC service architecture using SDN-assisted IP multicasting and SVC including its rationale. The remaining subsections describe the implementation of this service architecture.

4.3.1 Overview of the Proposed WebRTC Service Architecture

In the state of the art, each peer in a multi-party WebRTC session sends a scalable encoded video stream to a central SFU, which selectively forwards video layers to other peers according to their requirements. The main contribution of this work is to implement the functionality of the central SFU in a distributed fashion at SDN switches via SDN-assisted IP multicasting controlled by the SDN controller.

Rationale of the proposed architecture: The default multi-party WebRTC topology is mesh connection between peers, where limited upstream bandwidth of peers becomes a bottleneck. This problem is addressed by introducing the SFU service so that each peer sends a single upstream to the SFU. However, this requires a service provider to invest in high speed routers with high bandwidth connection since all video traffic must go through the SFU. This paper proposes SDN-assisted IP multicasting as a more efficient solution for delivery of managed WebRTC services using scalable encoded video, where video layers can be dropped at any SDN switch over the network (instead of at a central SFU) to adapt video resolution and rate according to terminal type of receiving peers, when WebRTC is offered as a service by an NSP or by a third-party video service provider (VSP) in collaboration with an NSP with SDN infrastructure. The proposed service architecture is beneficial for all parties involved: i) the peers get the lowest end-to-end delay with guaranteed desired downlink bandwidth according to their terminal type and they do not suffer from uplink bandwidth bottleneck, ii) the VSP can serve any number of customers without investing in expensive cloud SFU service, iii) SDN-assisted IP multicasting is more

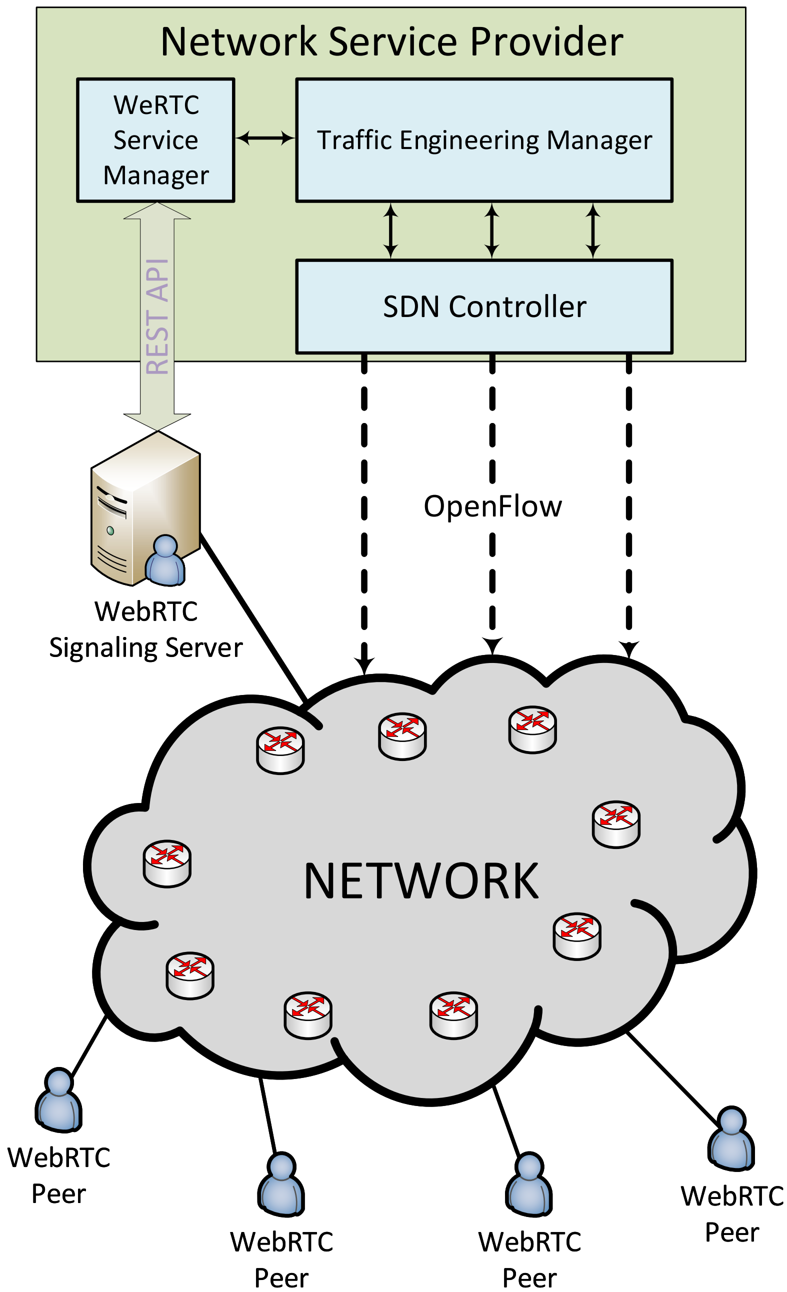

Figure 4.1: Architecture for SDN-assisted WebRTC service managed by the NSP.

efficient for the NSP in terms of overall network resource consumption compared to both central SFU and mesh service architectures and distributes traffic more evenly across the network.

An architecture that enables a VSP and an NSP having SDN infrastructure to collaborate in order to provision a managed WebRTC service is depicted in Figure 4.1. In this architecture, the NSP runs a WebRTC service manager (WSM) and a traffic engineering manager (TEM) as applications over the SDN controller. The VSP passes the IP addresses and terminal types of peers to the TEM via the WSM, and TEM computes the best multicast tree for each peer video stream and reserves end-to-end bandwidths accordingly. Main elements of the proposed managed WebRTC service architecture are introduced below.

WebRTC Signaling Server (WSS): The VSP runs a WSS anywhere in the network. Peers establish a session by exchanging Session Description Protocol (SDP) [21] and Internet Connectivity Establishment (ICE) [22] objects via the WSS. ICE objects contain IP address, transport protocol and port number of the peers. There is two-way interaction between the VSP and NSP by means of information exchange between the WSS and WSM. WSS passes peer information to the WSM, and receives path and bandwidth reservation confirmation from the WSM for appropriate peer billing. The details of the functionality and operation of WSS is explained in Section 4.3.2.

WebRTC Service Manager (WSM) is the NSP module that facilitates collaboration between the NSP and VSP to coordinate services, such as multicast tree construction, path/bandwidth reservation, and queue management and policing at switches, which the NSP provides to WebRTC peers. WSM gathers session information about peers from the WSS through REST API in order to create peer instances. WSM also communicates such service information back to the WSS.

Traffic Engineering Manager (TEM): TEM is an application that is connected to the northbound interface of the SDN controller of the NSP. The TEM constructs the lowest delay IP multicast tree subject to desired bandwidth constraints for video streams originating at each peer since the SDN controller has complete view of the network state including traffic load over all links and queue states. WSM module passes peer instance information to the TEM, which then computes a multicast tree for each peer video stream, reserves paths for receiving peers, and defines/updates

flow tables at the switches accordingly. This is explained in detail in Section 4.3.3. It also periodically allocates, updates, or destroys queues at the switches, and assigns WebRTC flows to the allocated queues. Details of these steps are discussed in Section 4.3.4.

In the proposed architecture, each SDN switch processes video packets at the transport and network layers in order to avoid deep packet inspection (DPI), which would be very costly. To this effect, we integrated the Multi Stream Transport (MST) framework [38], which uses different transport ports to send different scalability layers at the peers into the WebRTC source code [28]. Each peer encodes video with multiple spatial and temporal layers, where video packets have layer identifiers. Peers send different spatial and temporal video layers from different source ports. For example, if there are 3 spatial and 2 temporal layers, there will be $3\times2=6$ source ports. This enables efficient packet processing at the SDN switches to drop undesired video layers by checking only source port numbers. The details are explained in Section 4.3.5.

4.3.2 WebRTC Signaling Server and Multicast-Session Initiation

The main design principle of WebRTC is to generate matching PeerConnection object (PCO) pairs to simultaneously send and receive data between pairs of endpoints, i.e., each PCO establishes point-to-point communication with only one other PCO to send and receive data [28]. We need to make two major changes in the signaling server for session initiation to support IP-multicast in multi-party WebRTC service: i) Assuming there are N peers, we allow each peer to generate one "send-only" PCO that connects to "multiple" receive-only PCO (one at each receiving peer), and $N-1$ "receive-only" PCO, each connecting to a "send-only" PCO in another peer. ii) We employ a group key sharing protocol [39] between all peers since each video layer stream is multicast delivered to multiple peers in encrypted form.

The proposed WebRTC session initiation protocol that supports IP-multicast consists of two states, the SDP exchange state and ICE exchange state, which are summarized in Figure 4.2. First, the signaling server creates a separate list of PCO for

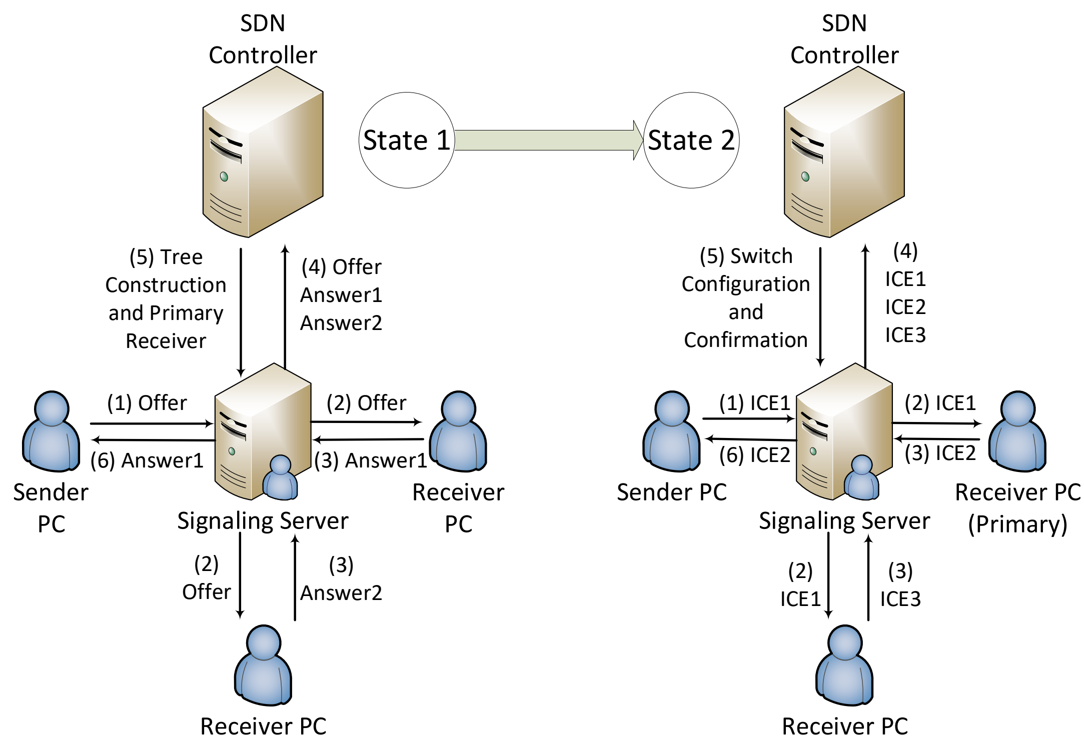

Figure 4.2: Proposed session initiation procedure at the Signaling Server.

each peer, which includes a send-only PCO and $N-1$ receive-only PCOs for each peer. Hence, a total of N lists with N PCO in each are created for a session.

The steps of SDP offer/answer exchange protocol are enumerated (1)-(6) in Figure 4.2. In step (1), one of the peers sends an offer type SDP with an "a=sendonly" line to the signaling server. The signaling server forwards this offer to all receive-only PC objects in its list in step (2). Then, in step (3) each receiver peer sends an answer type SDP, which contains the desired spatio-temporal layers to the signaling server. At this point, the signaling server passes offer type SDP and all answer type SDPs to the WSM module of the NSP. After processing the SDP objects, the WSM module passes the relevant information to the TEM module. TEM module orders peers according to decreasing number of video layers requested. The peers requesting the same number of layers are randomly ordered, although it is also possible to order them according to decreasing hop count from the sending peer. The first peer in the

ordered list is called the primary receiver. The TEM computes the multicast tree for this peer using Algorithm 3, and informs the signaling server about the primary receiver for the sending peer through the WSM module in step (5). Finally, in step (6) the signaling server passes the primary receiver's answer to the sender. This process is repeated for each sending peer in the session.

In the next state, ICE objects are exchanged between sender and receivers following the same 6 steps shown in Figure 4.2. That is, the sender passes its ICE object to all the receivers through the signaling server. Receivers send their ICE objects back. Every ICE object is passed to WSM module in order to inform controller about sessions. However, again only the primary receiver's ICE object is sent back to the sender. Finally, the controller configures switches according to constructed the multicast tree. Details of multicast tree construction at the TEM and port number usage/merging at the switches are discussed in the following subsections.

At session initiation, each sender transmits a STUN binding request to all of its $N-1$ receivers, and each receiver answers with a STUN binding response. These request packets are sent over the computed multicast trees. Each receiver gets this request and configures the application to receive stream state.

4.3.3 Multicast Tree Construction and Path Computation

In the best-effort WebRTC, media channels are formed along the shortest path between pairs of peers without considering congestion or delay. Since an SDN controller has full visibility of network traffic and switch queue states, an NSP with SDN infrastructure can perform centralized path computation at its TEM to route premium clients subscribed to managed services over service-specific network slices that satisfy specified bandwidth requirements with minimum delay. In the proposed NSP-managed WebRTC service architecture that utilizes SDN-assisted IP multicast, path computation and multicast tree construction are integrated. To this effect, we propose a new bandwidth and delay aware multicast tree construction (BDA-MTC) algorithm (see Algorithm 3). Using the BDA-MTC algorithm, the TEM successively

computes paths from a sender peer to all receiving peers ordered by decreasing number of video layers requested starting with the primary receiver, which requested the largest number of spatial and temporal layers, as defined in Section 4.3.2, given the IP address and transport port information of all peers. The BDA-MTC algorithm calls the bandwidth-constrained shortest-delay path finder (CSPF) algorithm, given in Algorithm 4, for path computation between endpoints. Note that a separate multicast tree is generated for each sender peer. Hence, we run Algorithm 3 once for each sender peer.

BDA-MTC Algorithm Description

Algorithm 3: Bandwidth and Delay Aware Multicast Tree

Input: G, $sender$, $receiver_{list}$, δ
Output: $path_{list}$
Sort $receiver_{list}$ according to decreasing number of requested spatial layers $r_i.SL$
Sort $receiver_{list}$ according to decreasing number of requested temporal layers $r_i.TL$ within each group of receivers with the same number of $r_i.SL$
$path_{list} \leftarrow (\emptyset, \emptyset)$ (Pairs of paths and destination peers)
foreach $r_i \in receiver_{list}$ **do**
  $r_i.bw \leftarrow$ Lookup.Table($r_i.SL$, $r_i.TL$)
  p^*, $delay^*$, $hc^* \leftarrow$ CSPF(G, $sender$, r_i, $r_i.bw$)
  $r_i.delay = delay^*$
  foreach *($peer_{dest}$, $path$)* $\in path_{list}$ **do**
    if *$r_i.SL$, $r_i.TL$ are compatible with $peer_{dest}$* **then**
      foreach $node \in path$ **do**
        $\sigma \leftarrow node$.delay (total delay between *sender* and *node* through multicast tree)
        p, $delay$, $hc \leftarrow$ CSPF(G, $node$, r_i, $r_i.bw$)
        if $hc < hc^*$ & $delay + \sigma \leq delay^* + \delta$ **then**
          $hc^* = hc$
          $p^* = p$
          $r_i.delay = delay + \sigma$
  $path_{list}$.add(r_i, p^*)

Algorithm 4: Constrained Shortest-Delay Path Finder

Input: G, $sender_i$, $receiver_i$, bw_i
Output: p_i^*, $delay_i$, hc_i
$p_i^* \leftarrow \emptyset$ (Set of links)
$delay_{p^*} \leftarrow$ Null
$hc_i = 0$
Create graph G^* such that $\forall$Link $\in G^* \geq bw_i$
$p_i^* \leftarrow$ Dijkstra(G^*, src_i, dst_i)
$delay_{p^*} =$ Sum of all latency delay over p_i^*
$hc_i =$ Number of links over p_i^*

The inputs to Algorithm 3 are a directed graph G representing the network topology, source peer *sender*, the list $receiver_{list}$ consisting of receiving peers r_i, and the parameter δ that can provide a trade off between service latency and total network resource consumption. The output is a list $path_{list}$ consisting of pairs of receiving peers r_i and the best path p^* from *sender* to r_i. The data structure r_i contains the desired number of spatial and temporal video layers, $r_i.SL$ and $r_i.TL$, respectively, bandwidth $r_i.bw$ and delay $r_i.delay$ values. The bandwidth and delay values are initialized as null at the beginning. The first path with the destination primary receiver starts at the sender peer. All other paths start at a switch along a path in $path_{list}$, where a multicast stream splits. Each path is represented by a data structure *path* that contains nodes *node* and links. The data structure *node* contains a value *node.delay* indicating the transmission delay between *sender* and node *node* through the multicast tree, which is periodically updated by the SDN controller.

In lines 3-4, the list $receiver_{list}$ is first sorted according to decreasing number of spatial layers ($r_i.SL$). Peers requesting the same number of $r_i.SL$ are then sorted according to decreasing number of temporal layers ($r_i.TL$).

In lines 6-19, we loop over the ordered list $receiver_{list}$ to construct the multicast tree starting with the path from *sender* to the primary receiver r_1.

In line 7, we set the bandwidth for receiver r_i depending on the desired number of spatial and temporal video layers according to a predefined lookup table.

In line 8, we call Algorithm 4 to find the shortest path p^* from *sender* to r_i

with associated delay $delay^*$ and hop count hc^*. The inputs to Algorithm 4 are a directed network graph G that consists of vertices (V) and edges (E), and the desired bandwidth $r_i.bw$ from the source peer *sender* to destination peer r_i. The CSPF algorithm creates a sub graph G^* by removing all links with bandwidth below $r_i.bw$. Then, it runs Dijkstra's Shortest Path algorithm [15] on G^*, where link delays are considered as cost and finds the path that satisfies required bandwidth $r_i.bw$ with the minimum delay $delay_{p^*}$.

For the primary receiver r_1, the path p^* computed in line 8 is the final path. The algorithm does not enter the loop in lines 10-18, since, initially there is no entry in the list $path_{list}$. Line 19 adds the primary receiver r_1 and the path p^* to $path_{list}$.

For the remaining receivers r_i, the algorithm enters the loop in lines 10-18 to search over all ($path$, $peer_{dest}$) pairs in $path_{list}$ for the best multicast split node for r_i that minimizes the total hop count hc^* (in order to minimize overall network resource consumption) subject to the delay constraint. Starting with the second receiver r_2, the algorithm loops over each *node* in *path* in lines 12-18 considering only paths currently in $path_{list}$ towards destination peers that requested equal or more number of spatial and temporal layers than r_i. In line 12, the delay between *sender* and the current node of *path* is defined as σ. In line 13, we call Algorithm 4 between end-points *node* and r_i. If the hop count hc is less than hc^* and $delay+\sigma$ is less than or equal to $delay^*+\delta$, then the output path p replaces p^* as the optimal path. Finally, r_i and the path p^*, which has the minimum hop count hc^* while satisfying the specified delay, are added to $path_{pair}$ list.

The proposed algorithm can generate a multicast tree minimizing either the transmission delay from *sender* to each receiver r_i or the total network resource consumption (related to the total hop count hc of the entire tree) or a function of both depending on the parameter δ. If we set $\delta = 0$, Algorithm 3 generates a multicast tree with the minimum delay paths from the *sender* to each receiving peer. On the other extreme, if we set $\delta = \infty$, Algorithm 3 finds a multicast tree that minimizes the total network resource consumption without taking delay into consideration. For

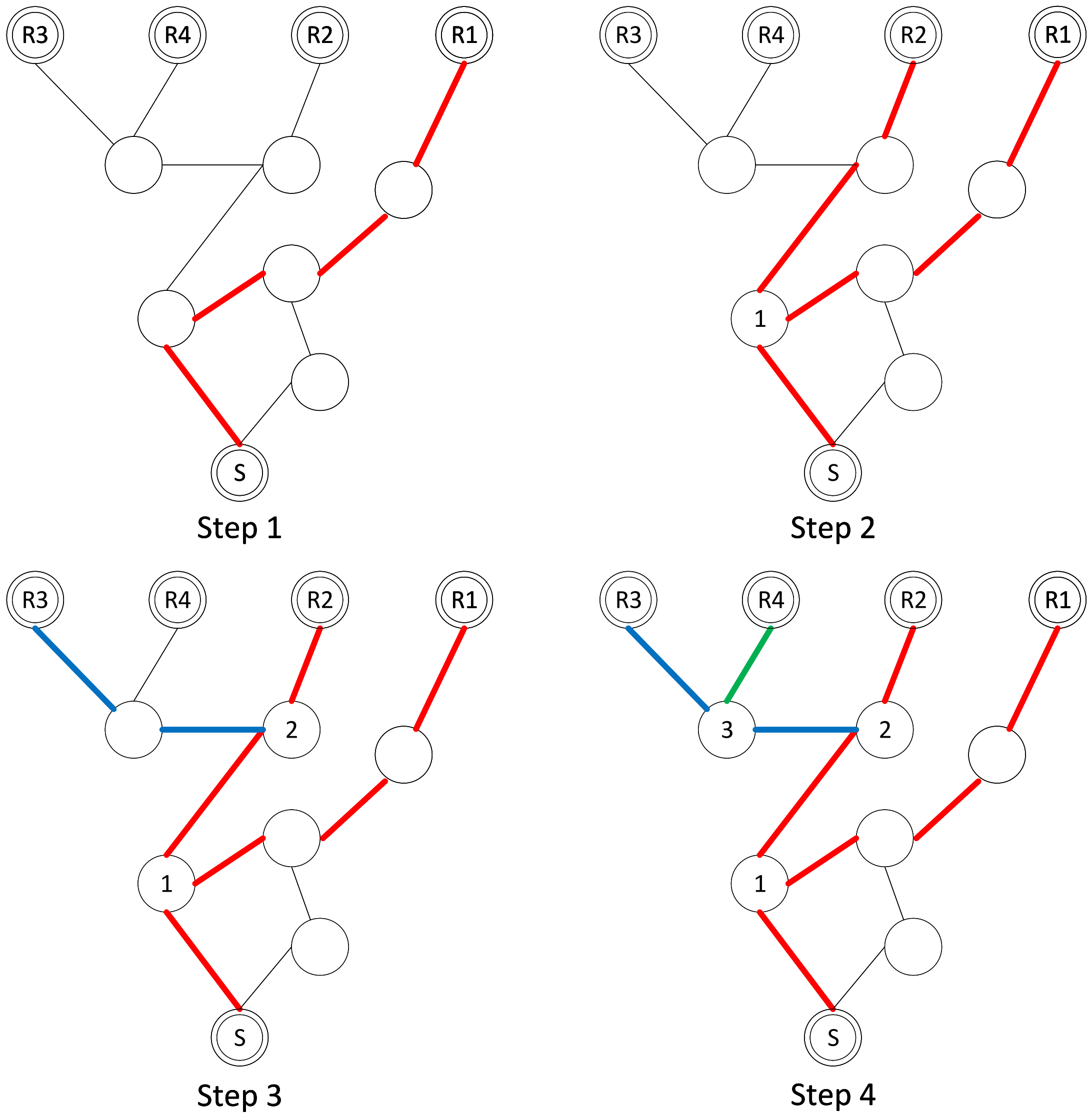

Figure 4.3: Demonstration of bandwidth and delay aware multicast tree generation.

other values $0 < \delta < \infty$, there is a trade-off between delay and total network resource consumption. Clearly, for the real-time communication application, where delay is very important, we recommend setting $\delta = 0$.

An Example

The operation of the proposed multicast tree construction algorithm is demonstrated by an example in Figure 4.3, where the sender s is connected to node S, and there

are four receivers $r1-r4$ connected to nodes $R1-R4$, respectively. We assume that receivers r1 and r2 request three layers (red), r3 two layers (blue), and r4 only one layer (green). In step 1, we compute the shortest delay path from S to R1, which is 4 hops from S. In Step 2, we search over all nodes along the red path to determine the best split node. Suppose that node 1 is identified as the best split node to reach R2. Similarly, in steps 3 and 4, we search over all nodes on all paths computed up that point to determine the best split nodes for R3 and R4, respectively. In the example, nodes 2 and 3 are found as the best split nodes to reach R3 and R4, respectively.

Complexity Analysis

In Algorithm 4, the complexity of creating the graph G^* with adjacency list data structure is $O(|E|+|V|)$ and running Dijkstra's shortest path algorithm with Fibonacci heap [15] is $O(Z)$, where $Z=|E|+|V|log|V|$. Therefore, the worst-case complexity of Algorithm 4 is $O(Z)$. Assuming there are C peers in a session, we sort $receiver_{list}$ two times in lines 3-4. The complexity of this part is $O(2ClogC)$. In lines 6-19, we run Algorithm 4 for each r_i and for each node in the paths $path_{list}$ and create a new graph. In the worst case, the number of nodes is equal to number of vertices $|V|$ in the graph, and Algorithm 4 costs $O(Z)$. We run Algorithm 3 C times, since, the number of senders is C in a single videoconferencing session. As a result, the total complexity to construct all multicast trees in a session is $O(C(2ClogC+C(Z+|V|(Z)))$ that is $O(C^2(logC+|V|Z))$. In videoconferencing, the number of peers C is typically a small constant compared to Z, $|V|$ and $|E|$, and each term in Z is smaller than $|E||V|$ or $|V|^2log|V|$. Hence, the total computational complexity can be approximated by $O(|E||V|+|V|^2log|V|)$.

4.3.4 Dynamic Resource Reservation by Queue Management

The TEM computes bandwidth and delay constrained multicast trees for all senders. Each multicast tree is defined by a data structure $path_{list}$. Resource reservation for all computed paths is managed by the SDN controller by directing managed WebRTC

flows to priority queues that are set up on all switches along all paths. A separate priority queue, whose resources are updated dynamically depending on the cumulative needs of all managed flows, is set up for all WebRTC traffic between each pair of switches.

In order to dynamically update the bitrate limit of a queue to accomodate all reserved WebRTC traffic between pairs of switches, we periodically create a new queue with the new target bitrate, transfer all existing flows in the old queue to this new queue, and then destroy the old queue.

4.3.5 Packet Processing at SDN Switches

The multicast trees defined by the set of $path_{list}$ are implemented at SDN switches under supervision of the SDN controller. The source node of each $path$ in $path_{list}$, except for the primary path, is the starting point of a branch of the multicast tree. Stream duplication and forwarding at these nodes are controlled by special action rules in the OpenFlow tables downloaded to these nodes by the SDN controller.

Packets are processed at the network and transport level at each SDN switch along each multicast tree. At each stream split point (switch), OpenFlow Groups are created as needed in order to duplicate the packets. An OpenFlow group consists of those receiving peers which receive the same scalability layers. Hence, only the requested layer streams are duplicated for each OpenFlow group. In addition, actions such as destination IP address modification, destination transport port number modification, and switch output ports setting are taken as needed for the both duplicated and actual packets. Packet processing for duplication of a subset of received streams for each OpenFlow group as necessary is depicted in Figure 4.4.

Receivers expect video packets with a single source port number as is agreed while exchanging ICE objects. Hence, layer streams with different source port numbers must be merged into a single stream before they are delivered to the receivers. At the final node, i.e., the node each receiving peer is connected to, streams with different source UDP port numbers are merged into a single stream with the agreed source port

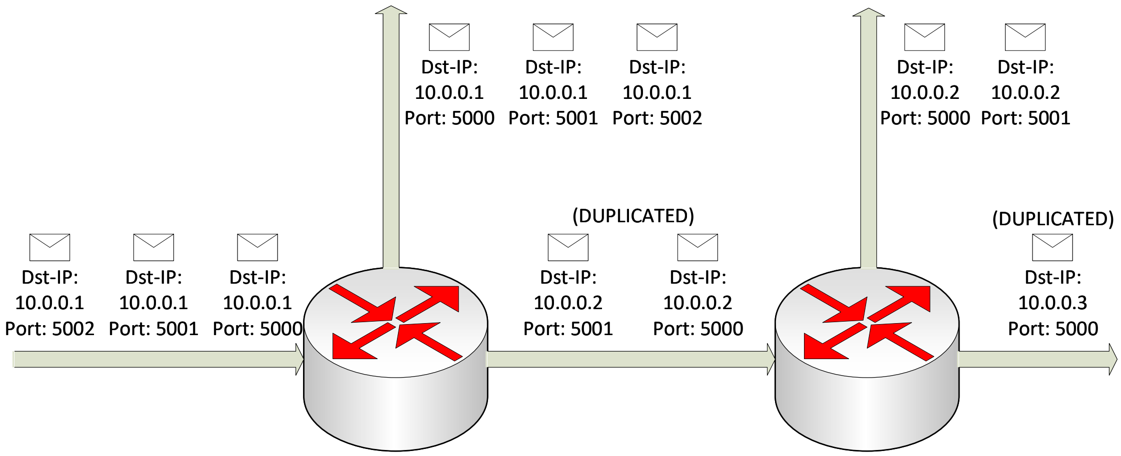

Figure 4.4: SDN-assisted multicasting of scalable video by selective video layer duplication and forwarding at SDN switches.

number. For example, suppose packets arriving into the last switch before a receiving peer have three different source port numbers, 33000, 33001 and 33002 representing three layers. This last switch merges them into a single stream with port number 33000 by the transport port modification action rule written by the SDN controller. As a result, the receiving peer receives all three layer packets as if they were sent as a single stream with a single UDP source port.

4.3.6 Peers

The peer applications consists of a sender side and a receiver side as depicted in Fig. 4.5. They are described below.

Sender Side

The sender side has a single *WebRTC Packetizer* and multiple *Asynchronous UDP Port* objects for sending different combinations of scalability layers.

The sender peer streams video considering the number of layers requested by each of the receiving peers. It creates a port for each unique spatio-temporal ID by incrementing the base port number. For instance, if the agreed source port number

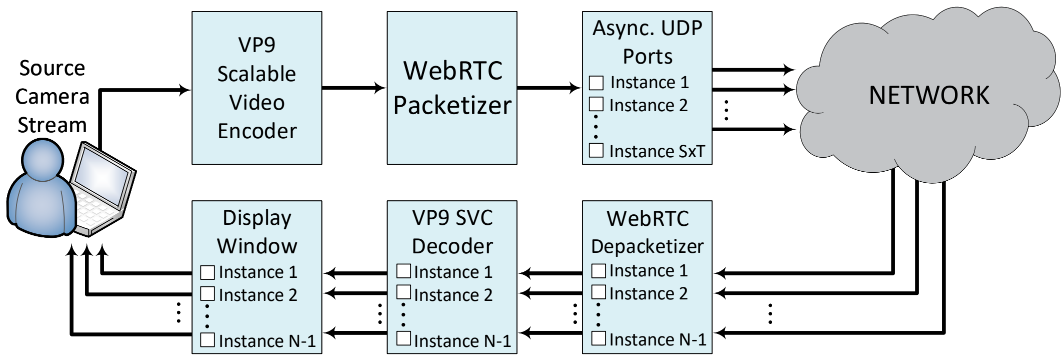

Figure 4.5: WebRTC peers for SDN-assisted IP multicasting using SVC codec.

of the sender is 33000, and there are 3 spatial layers (SL) and 2 temporal layers (TL). Client creates 6 asynchronous UDP sockets with the port numbers from 33000 to 33005. If we denote different combinations of spatial and temporal layers by SLiTLj, it sends SL0TL0 packets from 33000, SL1TL0 from 33001, SL2TL0 from 33002, SL0TL1 from 33003, SL1TL1 from 33004 and SL2TL1 from 33005. This layer-wise packet split in the transport layer with the source port number indicator enables Open vSwitches to differentiate, process and deliver various video layers without depacketizing at the application layer (RTP level).

Because we send streams over reserved end-to-end paths (or network slices), the target video bitrate is fixed and the RTCP feedback mechanism is disabled. This does not mean fixing the quantization parameter (QP), since for a target bitrate required QP may change depending on the video content.

In WebRTC, the sender sets a marker bit to indicate that the received spatial layer is decodable and the receiver should not wait for further enhancement layers. In the IP multicast service architecture, each receiver may desire different resolutions. Therefore, the sender peer does not set the marker bit for any spatial layer, but the receiver inserts the marker bit since it knows a priori which layers it will receive.

Receiver Side

The receiver side has separate *WebRTC Depacketizer* and *VP9 SVC Decoder* objects to process streams received from each *sender*, and separate *Display Window* objects to display them.

A receiver that receives all video layers gets RTP packets with successive Sequence Numbers without any loss, e.g., 1000, 1001, 1002, 1003. However, a receiver peer desiring lower spatial or temporal resolution video, gets Sequence Numbers with losses in between, e.g., 1000, 1001, 1004, 1005, because packets that belong to layers not requested are dropped at switches. Therefore, the NACK feedback mechanism is also disabled at receiving peers in addition to the RTCP feedback mechanism.

As stated before, the Marker Bit is set at the receiver side once all requested layers of a frame are received.

4.4 Evaluation

This section compares the performance of the proposed SDN-assisted IP multicast WebRTC service architecture with those of the standard mesh- and SFU-connected services. We describe an emulation environment and a simulation set-up in Section 4.4.1. Experimental simulation results for comparative performance evaluation are presented in Section 4.4.2.

4.4.1 Experimental Set-Up

We first implemented the proposed SDN-assisted IP multicast WebRTC service architecture over an emulation environment, which is described in Section 4.4.1, as a proof of concept that the proposed architecture successfully runs over real network protocols using WebRTC signaling server and native WebRTC peer applications. We also implemented a simulator in order to evaluate the large scale network performance of the mesh, SFU and SDN-assisted IP multicast service architectures. The simulation environment is described in Section 4.4.1.

Emulation Environment

We emulated a network with 20 virtual (OVS) switches using the Mininet [17] running on a server. The Floodlight SDN controller [16] runs as a separate application on the same server. The SDN controller is connected to the Mininet through a specified port and talks to the switches through its soutbound interface. Our implementation is OpenFlow 1.5 compliant. It can also be implemented on P4 enabled switches [40]. The WebRTC signaling server runs as an application in one of the virtual hosts that is connected to the Mininet. We connected three real hosts (laptops) to the Mininet virtual switches using ethernet to USB hardware interfaces (hw-intf) in order to run WebRTC applications with VP9 encoders. The emulation environment is depicted in Figure 4.6. We prefered to run each WebRTC peer application on a separate personal computer, since running the Mininet, SDN controller, WebRTC signaling server, and multiple instances of VP9 encoder/decoder on the same server causes real-time CPU performance limitations.

Each client runs a scalable video encoder configured to encode video with two spatial layers, 360p base layer at 350 kbps and 720p enhancement layer at a total bitrate of 1 Mbps. All clients with terminal type HD receives both layers and clients with SD terminal receive only one layer video.

Simulation Environment

Since it is difficult to implement an emulation environment with hundreds of hosts, we implemented a simulation environment to evaluate the performance of the service architectures in large scale networks with a large number of hosts (sessions). To this effect we simulated networks with 256, 400, ..., 1600 switches with random topology. In order to model physical distances between switches, we consider a rectangular grid of unit squares for each network size, where there are as many squares as the number of switches. Each switch is placed at a random coordinate within a unit square.

We employ the physical distance based probabilistic model proposed in [24] to create the links, where the probability P of creating a link between two nodes is given

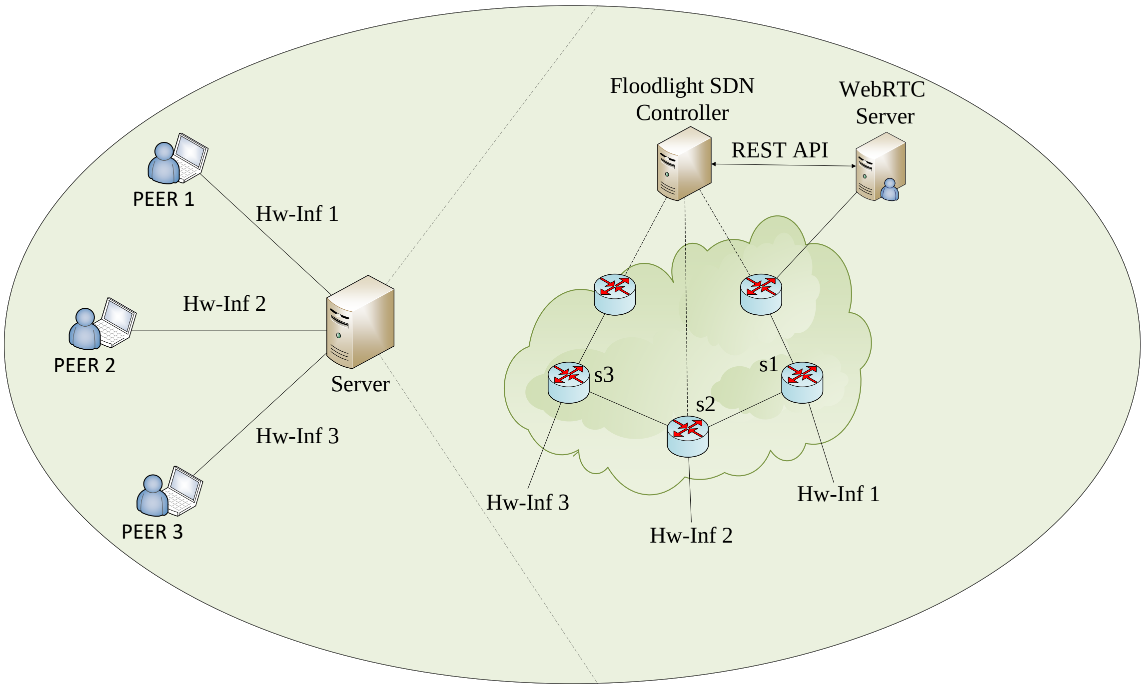

Figure 4.6: Test environment for multi-party WebRTC videoconferencing experiments over the Mininet with Floodlight Controller.

by $\mathrm{P} = \beta \exp\left(\frac{-d}{L\alpha}\right)$ where d is the Euclidean distance between the nodes, L is the maximum Euclidian distance between any two nodes, and parameters α and β are chosen by the user. We generate bidirectional links between pairs of nodes that are connected. An example random graph with 100 nodes and ≈ 450 bidirectional links is shown in Figure 4.7, where α is 0.05 and β is 5.

In the simulations, we assume there is no congestion across links and each link introduces a fixed delay of 5 msec., since the service is delivered over paths reserved by the SDN controller of the NSP. The delay value is determined as the average of Traceroute measurements [25] from our university to different university servers across the globe.

For each network size, we have 3 experimental scenarios, where in each scenario there are 100 service requests (sessions) with 3, 4, and 5 peers in each session, respectively.

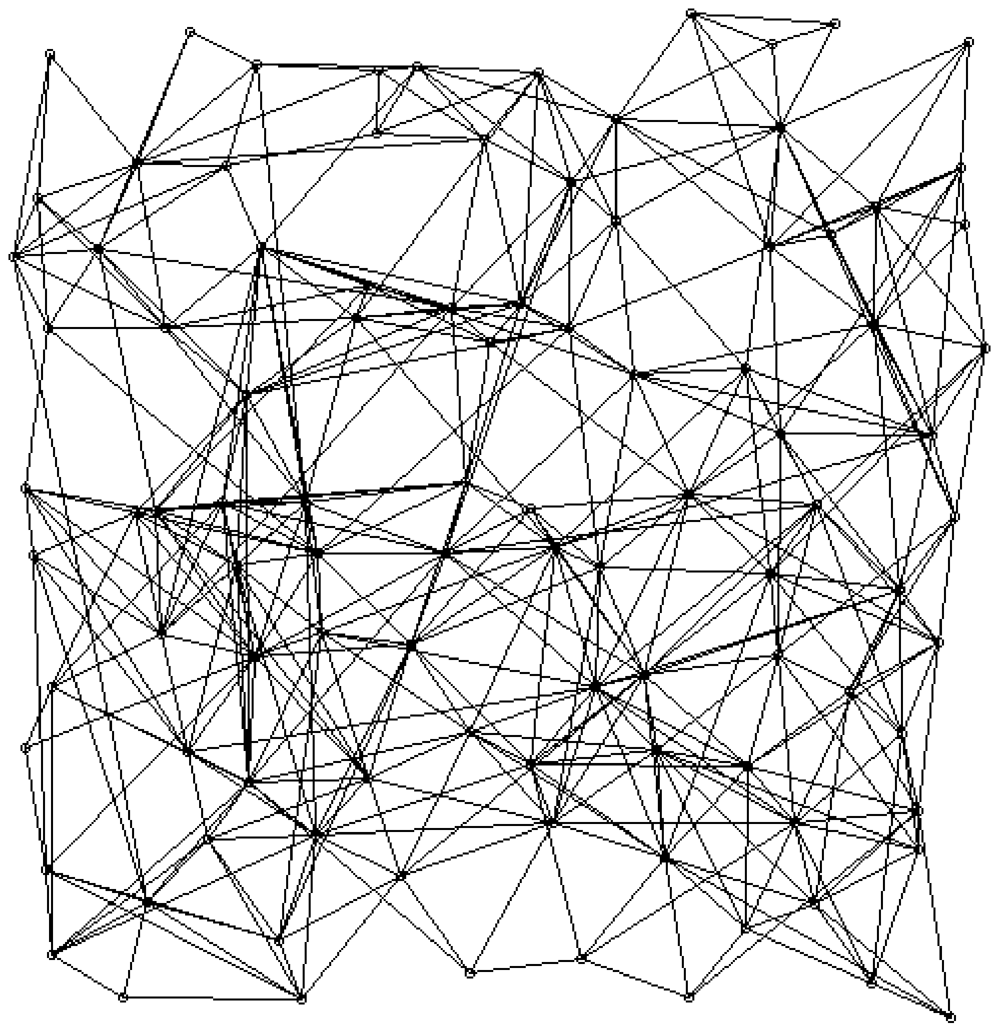

Figure 4.7: A realization of random graph topology.

4.4.2 Experimental Results

We compare the performance of the proposed SDN-assisted IP multicast service architecture (for $\delta = \infty$ and $\delta = 0$) with those of the mesh and SFU architectures based on four measures: average service delay, maximum service delay, total overall network resource consumption, and maximum link load. Average delay is the mean over 100 realizations of the average delay in 100 sessions in one realization. Maximum delay considers the mean over 100 realizations of the peak service delay among 100 sessions

in each realization. Total network resource consumption refers to sum of network capacity (bandwidth) consumed by all peers on all links. Maximum link load is the maximum of all individual link loads generated by the service. These measures aim to quantify: i) the average end-to-end service latency, ii) the worst-case end-to-end service latency, iii) the efficiency of each architecture in terms of total network resource consumption, and iv) whether a service creates congestion in any part of the network.

Network Topology and Size, Number of Sessions and Peers: We run three different experiments, where we have 100 sessions with 3, 4 and 5 peers in each session, in the experiment 1, 2 and 3, respectively. We repeat each experiment with different number of switches in the network ranging from 256 to 1600 switches according to the formula $(4n)^2$ where $n = 4, ..., 10$. We run 100 trials of each of experiments 1, 2 and 3 for each network size with different realizations of random topology graphs (random switch positions and links), peers connected to random switches, random location of the SFU server (if present), and the terminal types of peers selected randomly. The switch number that each peer and the SFU server is connected is chosen according to uniform PMF; i.e., we draw a uniform random number between 0 and 1, and multiply this number with the total number of switches to determine a randomly selected switch. We compute our four measures for each repetition of each experiment using the mesh, central SFU, and the proposed SDN-assisted IP multicast architectures on the same network graph. We report the average and the standard deviation of these measures over 100 trials.

Peer Video Display Resolutions: We picked the percentage of peer device types (received video resolution) according to the numbers projected for 2021 in [26], which estimated the distribution of device video resolutions as 41%, 47% and 12% for SD, HD, and UHD, respectively. Since we considered 720p and 360p video in our experiments, we set the probability that a peer requests 360p video as 0.4 and 720p video as 0.6.

Test Videos: We used five 720p raw test videos (with .yuv extension), namely

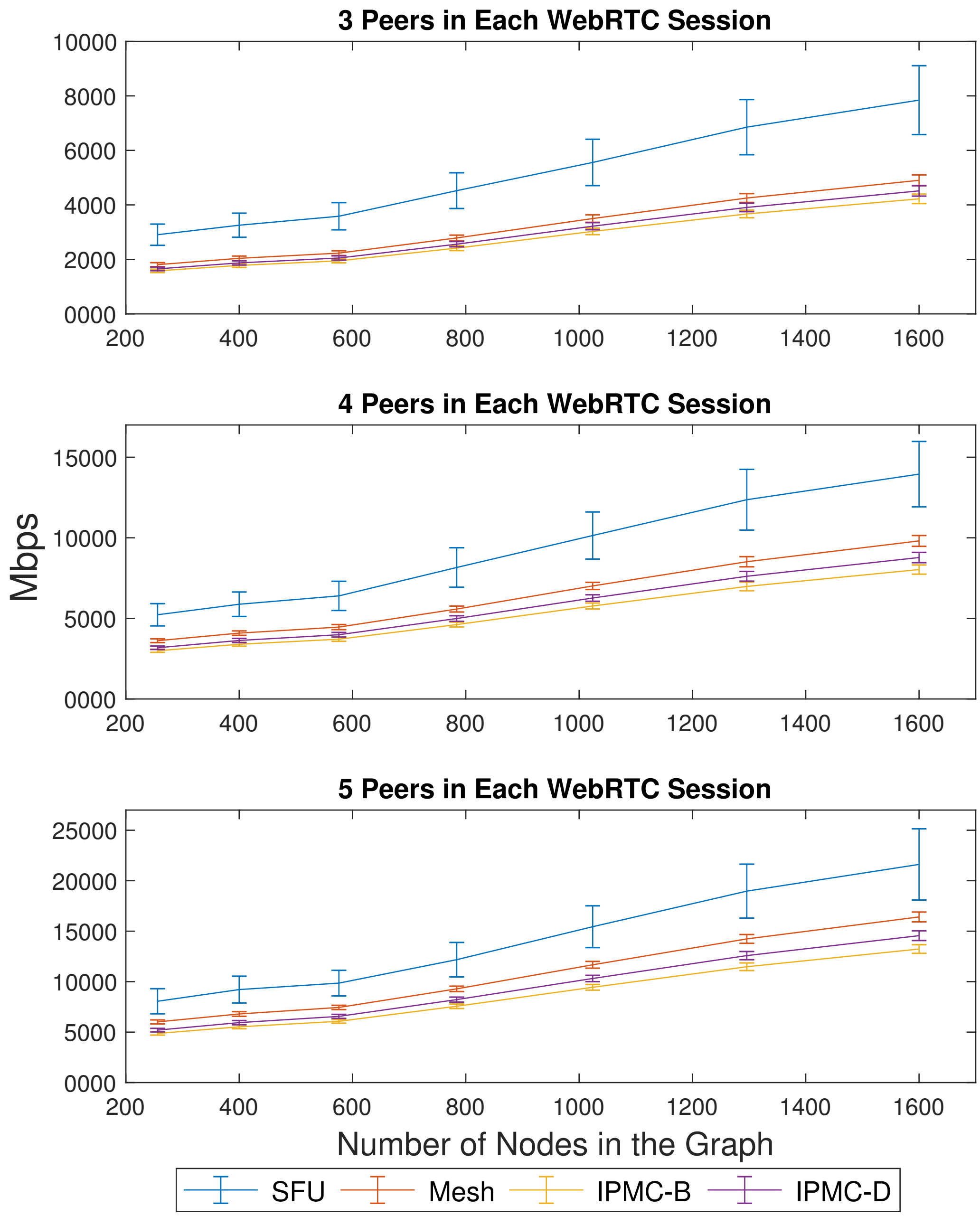

Figure 4.8: Mean and standard deviation of the total network resource consumption in 100 trials of 100 services with 3, 4, 5 peers for different size networks.

FourPeople (FP), Johnny, KristenAndSara (K&A), Vidyo3, Vidyo4 that are available in [27]. There is no global camera motion or large motion in these videos, which

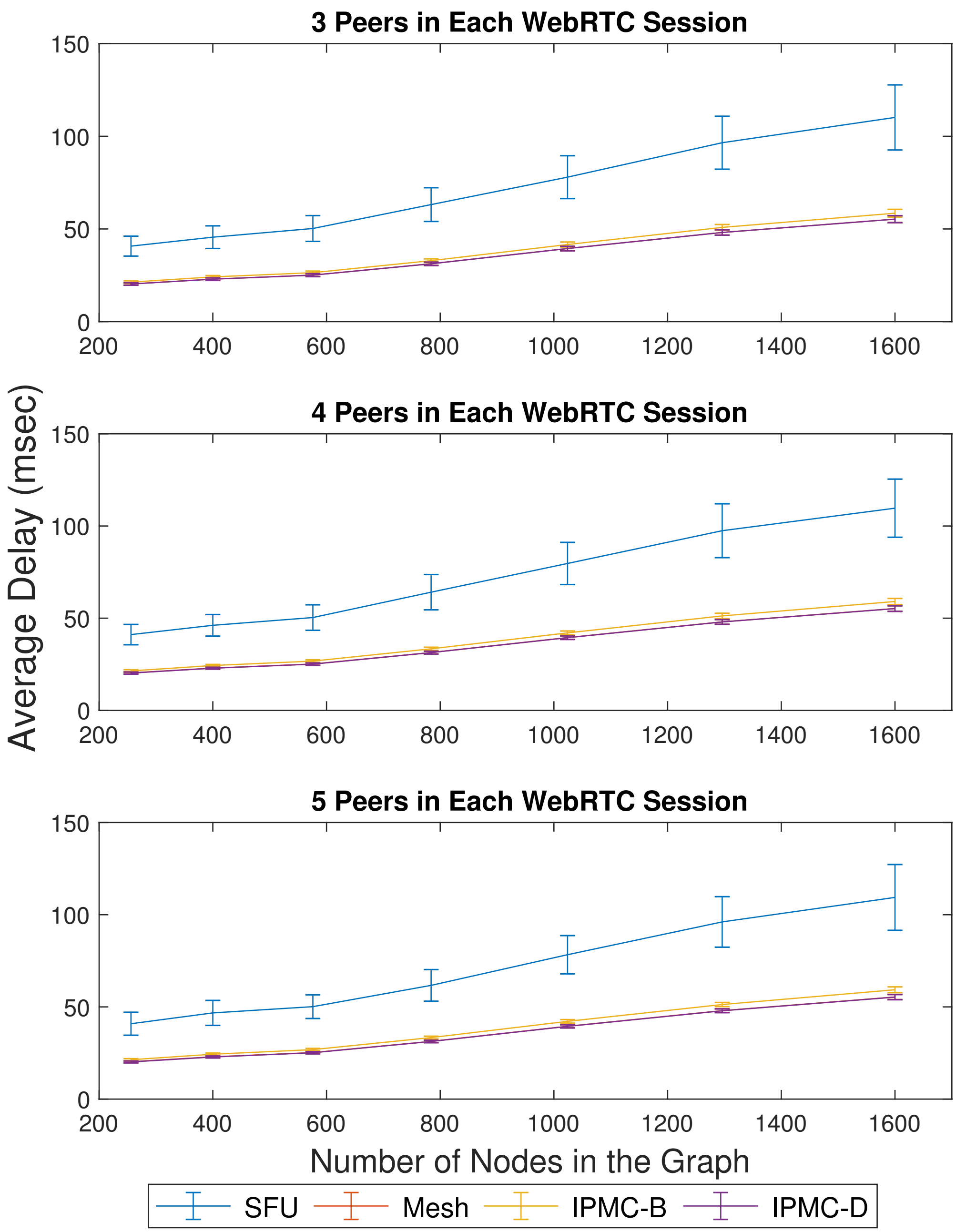

Figure 4.9: Mean and standard deviation of the mean E2E delay in 100 trials of 100 services with 3, 4, 5 peers for different size networks.

is typical of a real-time videoconferencing session. We encoded these videos using scalable VP9 with two spatial layers, where base layer is 360p encoded at 350 kbps and

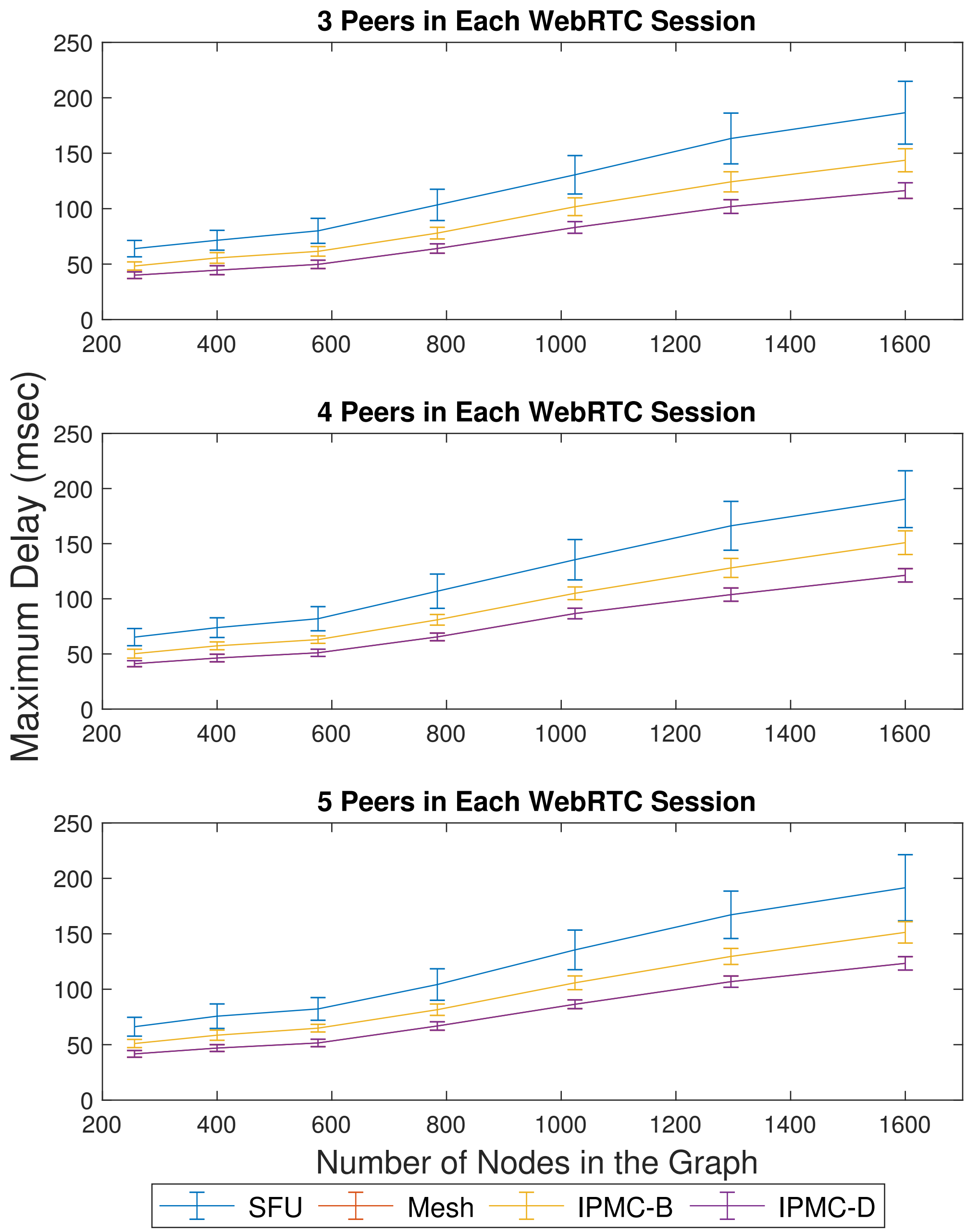

Figure 4.10: Mean and standard deviation of the maximum E2E delay in 100 trials of 100 services with 3, 4, 5 peers for different size networks.

enhancement layer is 720p encoded at 1 Mbps (including the base layer). The PSNR values of the encoded videos at these bitrates are shown in Table 4.1 that indicate

Table 4.1: PSNR(dB) of scalable coded videos according to spatial layer.

	FP	Johnny	K&A	Vidyo3	Vidyo4
720p	37.59	38.82	38.08	36.45	37.13
360p	33.30	35.92	33.37	31.61	33.19

good quality. Since the service is delivered over paths with reserved bandwidth, the received videos will have the same PSNR shown in Table 4.1.

Service Topologies Compared: We compare the performance of the proposed SDN-assisted IP multicast service architecture optimized to minimize delay (IPMC-D) running Algorithm 3 with $\delta = 0$, and optimized to minimize total network resources consumed (IPMC-B) running Algorithm 3 with $\delta = \infty$, against the performances of the state of the art mesh- and SFU-connected service architectures using our four measures. The results are discussed below:

Mean E2E Service Delay: The mean and standard deviation of the average delay of 100 sessions as a function of network size are depicted in Figure 4.9. SFU method has the largest mean service delay whereas the mesh and IPMC-D (with $\delta = 0$) methods have the smallest mean service delay. We note that mesh and IPMC-D have identical mean service delay values. IPMC-B method has a slightly larger average service delay compared to IPMC-D and mesh architectures. As the number of peers in a session increases, the service delay gap between the SFU method and the other three grows larger. We observe that the mean service delay achieved by the proposed, IPMC-D method is under 50 msec in all cases, which is very good.

Maximum E2E Service Delay: The mean and standard deviation over 100 realizations of the maximum delay of all peers in 100 sessions, which quantifies the worst-case latency, as a function of network size is shown in Figure 4.10. The SFU method has the largest worst-case latency, whereas the mesh and IPMC-D methods have the smallest. The maximum delay obtained by the IPMC-B method is slightly larger than that of IPMC-D and mesh. As the number of peers in a session increases, the difference between the worst-case performances of the SFU method and the other

Table 4.2: The load of link with the maximum load in Mbps

	SFU	Mesh	IPMC-B	IPMC-D
3 peers	206.63	16.32	14.47	15.08
4 peers	416.32	30.90	26.09	27.85
5 peers	683.79	49.54	41.61	44.66

three grows larger. It is important to observe that the maximum E2E service latency remains under 100 msec. only in the case of the mesh and the proposed IPMC-D methods.

Total Network Resource Consumption: This measure quantifies the efficiency of service architecture in terms of network resource usage, which is of interest to the NSP. The total network resource consumption of different methods as a function of the network size is depicted in Figure 4.8. We see that the SFU method consumes the highest amount of resources by a large gap in all cases. Both IPMC-B and IPMC-D require the smallest total network resources to deliver the service. As the number of peers in a session increases, the mesh method consumes more resources, while the resources consumed by the IPMC-B and IPMC-D methods grows at a slower rate.

Maximum Link Load: This measure indicates whether any of the service architectures creates congestion bottleneck in any part of the network. The maximum link loads for each service architecture are shown in Table 4.2. The results indicate that unlike other methods the SFU method has the potential to create network congestion in the vicinity of the SFU server since all traffic must go through the SFU server.

Overall, the analysis of experimental results show that the mean and standard deviation of the average service latency, the maximum service latency, overall network resource usage, the maximum link load of the proposed service architecture is much lower than the state of the art SFU service.

4.5 Conclusion

This paper proposes a new NSP-managed SDN-assisted delay and bandwidth aware IP multicast WebRTC service architecture that can be deployed as a value-added service over future 5G networks as a better alternative to the mesh- or SFU-connected service architectures that are in use today.

SDN-assisted IP multicasting of scalable encoded video can be considered as selective duplication and forwarding of video layers at network switches over the entire network in a distributed manner under the supervision of the SDN controller as opposed to selective forwarding at a central SFU server. Our extensive experimental results demonstrate that the proposed service architecture satisfies all requirements, including lowest average and maximum service latency and single upstream from peers, as well as it is the most efficient in terms of overall network load (resource consumption) and distributes the traffic load more evenly over the network.

Our results clearly show that the state of the art SFU-based real-time communication services have poor end-to-end service latency and overall network resource usage performances. The SFU service also has potential to create congestion in the vicinity of the SFU since all traffic must go through the SFU. The proposed NSP-managed SDN-assisted delay and bandwidth aware IP multicast service architecture solves all of these problems.

Chapter 5

CONCLUSION

This thesis addresses multi-party WebRTC videoconferencing architecture problems gradually starting from best-effort over-the-top to managed value-added services. Integrating multi-party WebRTC services over 5G infrastructure for better quality of service and quality of experience is one major contribution. Proposing novel SDN-assisted multi-party WebRTC streaming architectures and improving existing state of the art architectures is the other important contribution.

Best-effort OTT video services suffer from variant network conditions for all type of streaming architectures. System architectures of mesh, SFU and IP Multicast video services with the advance of 5G network infrastructure are proposed in Chapter 2, Chapter 3 and Chapter 4 respectively. End-to-end network slicing for video services guarantees streaming bitrate capacity and enables traffic resilient video services.

For mesh architecture, both motion-adaptive best effort and managed services in Chapter 2 have better video quality results comparing to default services. It is clearly seen that the peer with least receiving capacity becomes limiting factor for the rest of the peers in default implementation. Either proposed solution addresses this problem, moreover, SDN-assisted managed video services has the best quality results. Nonetheless, mesh connection still has limited uplink capacity in default and proposed architectures.

State of the art SFU architecture addresses limited uplink capacity problem of the mesh connected multi-party WebRTC architecture, yet, it introduces high amount of total network bandwidth consumption and extra end-to-end delay. SDN-assisted distributed edge-SFU is a novel WebRTC videoconferencing streaming architecture that uses multiple SFU servers for a single videoconferencing session. This service

architecture reduces total network resource consumption and end-to-end service delay while solving limited uplink capacity problem of mesh architecture which is discussed in detail in Chapter 3. Large scale simulation results show that distributed edge-SFU is superior to state of the art random anywhere-SFU architecture.

NSP-managed SDN-assisted delay and bandiwdth aware IP multicast WebRTC service architecture over 5G network infrastructure is proposed in Chapter 4. Experimental results show that this architecture satisfies all requirements including lowest avareage and maximum service latency, single upstream from peers for limited uplink capacity, and the most efficient total network resource consumption.

BIBLIOGRAPHY

[1] R.A. Kirmizioglu, B.C. Kaya, and A.M. Tekalp, "Multi-party WebRTC videoconferencing using scalable VP9 video: From best-effort over-the-top to managed value-added services," IEEE Int. Conf. Multimedia and Expo (ICME), San Diego, CA, USA, July 2018.

[2] R.A Kirmizioglu and A.M. Tekalp, "Managed Multi-party WebRTC Service using SDN-Assisted IP Multicasting and SVC," IEEE Trans. on Multimedia.

[3] R.A Kirmizioglu, A.M. Tekalp and B. Gorkemli, "Optimization of Multi-Party WebRTC Videoconferencing Service by SDN-assisted Edge Computing," IEEE Trans. on Multimedia.

[4] H. Alvestrand, "Overview: Real Time Protocols for Browser-based Applications," draft-ietf-rtcweb-overview-19, Nov. 12, 2017.

[5] G. Carlucci, L. De Cicco, S. Holmer, and S. Mascolo, “Analysis and design of the Google congestion control for web real-time communication (WebRTC)," Proc. 7th Int. Conf. on Multimedia Systems, 2016.

[6] A. Grange, P. de Rivaz, and J. Hunt, VP9 Bitstream and Decoding Process Specification, 31 March 2016.

[7] J. De Cock, A. Mavlankar, A. Moorthy, and A. Aaron, "A large-scale video codec comparison of x264, x265 and libvpx for practical VOD applications," SPIE Proceedings Vol. 9971, Applications of Digital Image Processing XXXIX, San Diego, CA, Sep. 2016.

[8] “Software-defined networking: The new norm for networks,” Open Networking Foundation (ONF) White Paper, 2012.

[9] A. Eleftheriadis, M.R. Civanlar, and O. Shapiro, “Multipoint videoconferencing with scalable video coding," Jou. of Zhejiang University Science A, vol. 7, no. 5, pp. 696-705, 2006.

[10] G. Bakar, R. A. Kirmizioglu, and A. M. Tekalp, "Motion-based adaptive streaming in WebRTC using spatio-temporal scalable VP9 video coding," IEEE Globecom, Singapore, Dec. 2017.

[11] K.-F. Ng, M.-Y. Ching, Y. Liu, T. Cai, L. Li, and W. Chou, "A P2P-MCU approach to multi-Party video conference with WebRTC," Int. Jou. of Future Computer and Communication, vol. 3, no. 5, Oct. 2014.

[12] S. Yoon, T. Na, and H.-Y. Ryu, "An implementation of Web-RTC based audio/video conferencing system on virtualized cloud," IEEE Int. Conf. on Consumer Electronics (ICCE), 2016.

[13] A. Boubendir, E. Bertin, and N. Simoni, "Network as-a-Service: the WebRTC case: How SDN and NFV set a solid Telco-OTT groundwork," Int. Conf. on the Network of the Future (NOF), Montreal, Canada, 2015.

[14] S. Jero, V. K. Gurbani, R. Miller, B. Cilli, C. Payette and S. Sharma, "Dynamic control of real-time communication (RTC) using SDN: A case study of a 5G end-to-end service," Proc. of IEEE/IFIP Network Operations and Management Symposium (NOMS), 2016.

[15] E. W. Dijkstra "A note on two problems in connexion with graphs," Numerische Mathematik 1, 269-271.

[16] Project Floodlight Open-Source SDN Controller http://www.projectfloodlight.org/floodlight/

[17] Mininet Virtual Network http://mininet.org/

[18] https://www.opennetworking.org/seba/

[19] VP9 video codec implementation. https://www.webmproject.org/vp9/

[20] K.T. Bagci, K.E. Sahin, and A.M. Tekalp, "Compete or collaborate: Architectures for collaborative DASH video over future networks," IEEE Trans. on Multimedia, vol. 19, no. 10, pp. 2152-2165.

[21] M. Handley, "SDP: Session Description Protocol," RFC 4566, July 2006.

[22] J. Rosenberg, "Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal for Offer/Answer Protocols," RFC 5245, Apr. 2010.

[23] Janus Gateway, online https://github.com/meetecho/janus-gateway

[24] E. Zegura, K. Calvert, and S. Bhattacharjee, "How to model an internetwork," in Proc. IEEE Infocom, vol. 2, March 1996, pp. 594-602.

[25] G. Malkin, "Traceroute Using an IP Option," RFC 1393, Jan. 1993.

[26] "The Zettabyte Era: Trends and Analysis, Cisco Whitepaper," June 2017. Available Online.

[27] Test videos https://media.xiph.org/video/derf/

[28] WebRTC native code https://webrtc.org/native-code/

[29] I. F. Akyildiz, A. Lee, P. Wang, M. Luo, and W. Chou, "A roadmap for traffic engineering in SDN-OpenFlow networks," Computer Networks, vol. 71, pp. 1–30, 2014.

[30] M. Karakus and A. Durresi, "Quality of service (QoS) in software defined networking (SDN): A survey," Journal of Network and Computer Applications, 2016.

[31] K.T. Bagci, K.E. Sahin, and A. M. Tekalp, "Compete or collaborate: Architectures for collaborative DASH video over future networks," IEEE Trans. on Multimedia, vol. 19, no. 10, pp. 2152-2165, October 2017.

[32] S. McCanne, V. Jacobson, and M. Vetterli, "Receiver-driven layered multicast," ACM SIGCOMM, pp. 117–130, August 1996.

[33] Q. Zhang, Q. Guo, Q. Ni, W. Zhu, and Y.-Q. Zhang, "Sender-adaptive and receiver-driven layered multicast for scalable video over the Internet," IEEE Trans. Circ. Syst. for Video Tech., vol. 15, no. 4, pp. 482-495, April 2005.

[34] J.-P. Sheu, C.-W. Chang, and Y.-C. Chang, "Efficient multicast algorithms for scalable video coding in software-defined networking," IEEE Int. Symp. on PIMRC, Hong Kong, 2015.

[35] S. Shen, L. Huang, D. Yang, and W. Chen, "Reliable multicast routing for software-defined networks," Proc. IEEE INFOCOM, 2015, pp. 1–9.

[36] J. Yang, E. Yang, Y. Ran, Y. Bi, and J. Wang, "Controllable multicast for adaptive scalable video streaming in software-defined networks," IEEE Trans. on Multimedia, vol. 20, no. 5, pp. 1260-1274, May 2018.

[37] N. Xue, X. Chen, L. Gong, S. Li, D. Hu, and Z. Zhu, "Demonstration of OpenFlow-controlled network orchestration for adaptive SVC video manycast," IEEE Trans. on Multimedia, vol. 17, no. 9, pp. 1617-1629, Sept. 2015.

[38] S. Wenger, "RTP Payload Format for Scalable Video Coding," RFC 6190, May 2011.

[39] M. Baugher, et al., “Multicast Security (MSEC) Group Key Management Architecture," RFC 4046, April 2005.

[40] https://p4.org/